COMMODORE 64

INTRODUZIONE AL BASIC. PARTE 2

COD. 6025-I

LA SERIE COMPLETA DI PROGRAMMAZIONE AUTODIDATTICA

Questo corso è la Parte 2 di una serie intesa ad aiutare a conoscere ogni aspetto di programmazione del computer Commodore **64**. Esso parte dai principi illustrati nella Parte 1 per fornire tutta la conoscenza necessaria per scrivere programmi BASIC ben strutturati sul computer **64**. Il corso è costituito da due parti:

1. Un testo autodidattico diviso in 11 lezioni o "Unità", ciascuna delle quali tratta un importante aspetto della programmazione.
2. Due cassette di nastro che contengono una raccolta di programmi **64**, che aiuteranno a studiare le unità.

Inoltre viene fornita una squadretta per aiutare il disegno degli schemi di flusso.

INDICE

Titolo	Oggetto	Programmi relativi su cassetta	Pagina
	Introduzione		
Unità 16	Le funzioni DATA, READ e RESTORE: Uso di una iterazione; le istruzioni DATA e READ; formato dalle istruzioni DATA; errori delle istruzioni DATA e READ; il comando RESTORE	UNIT16QUIZ	157
Unità 17	Come affrontare la complessità dei programmi: uso del due punti; uso delle istruzioni IF-THEN; operatori logici; l'operatore AND; l'operatore OR; combinazioni di operatori logici; il comando NOT.	UNIT17PROGR	165
Unità 18	Introduzione delle subroutine: formato della subroutine; come funziona GOSUB; trasmissione di parametri a e da subroutine	PICTURE	175
Unità 19	Altro sulle subroutine: specifica delle subroutine; esempio di subroutine per semplificare le frazioni; robustezza delle subroutine; limitazione del campo di un parametro; convenzioni di denominazione	BIGLETTERS	185
Unità 20	Matrici: l'istruzione DIMENSION; uso delle variabili matrici; ulteriore uso delle matrici con le istruzioni DATA	UNIT20QUIZ	195
Unità 21	Funzioni stringa: la funzione LEN; la funzione MID$; estrazione di cognomi; uso di MID$ per correggere una stringa; le funzioni LEFT$ e RIGHT$; permutazioni; rimozione di lettere da una stringa; conversione di stringhe in numeri con l'uso di VAL; come evitare l'eccedenza delle parole sullo schermo	UNIT21QUIZ	203

Titolo	Oggetto	Programmi relativi su cassetta	Pagina
Unità 22	Uso delle matrici per ricercare e riordinare; la ricerca dicotomica; il Bubble Sort; il Quicksort; confronto dei tempi di riordino; la funzione FRE; esaurimento dello spazio di memoria del **64** matrici bidimensionali	QUICKSORT LIFESTART	219
Unità 23	Uno sguardo ravvicinato all'interno del **64** organizzazione della memoria del **64** byte; il comando PEEK, il comando POKE; introduzione all'animazione; altro su PEEK e POKE; esempio di animazione	WASPS	233
Unità 24	Argomenti vari: altro sugli operatori logici; come il **64** valuta le condizioni; codici ASCII CBM, la funzione ASC; il comando ON; il comando END; il comando DEF; memorizzazione e richiamo di dati su cassetta; il comando PRINT ; il comando INPUT ; il comando GET	MAKENAMES	253
Unità 25	Disegno di programmi - casi esemplificativi: frasi casuali, giochi di avventura o labirinto	GRAFFS, DUNGEON	273
Unità 26	Gli SPRITES: il disegno degli Sprites; inserimento degli Sprites nei programmi; il controllo degli Sprites; studio dei programmi degli Sprites; Sprites multicolori, gli schermi con molti Sprites.	SPRITE DISPLAY PROGRAM (SDP) SPRITE EDITOR (MONSPR) LINEAR, CIRCULAR 1 CIRCULAR 2 COUNCE, BUS COSPRED, SOLAR, SPRMAC GROTTY, PLANETS	287
Conclusione			307
Append. A	Creazione del suono col Commdore **64**	DUBLIN TUNE PLAYER SHEBA	309
Append. B	Libreria di subroutine	LIBRARY	318
Append. C	Risposte degli esperimenti	Esp. 23.3A (T o D) Esp. 24.3B (T o D) DATEPROG (T o D)	326
Indice alfabetico			336

INTRODUZIONE

Benvenuti alla seconda parte del corso **64** BASIC. La struttura del libro e i nastri saranno già noti dato che il corso è una continuazione diretta del **64** BASIC Parte 1. Le Unità sono state numerate consecutivamente dal 16 in avanti per sottolineare questa continuità.
Innanzitutto si troveranno le Unità abbastanza simili a quelle già studiate ma procedendo nel manuale diventeranno sempre più lunghe e probabilmente un pochino più difficili. Ci si occuperà delle applicazioni avanzate del BASIC, comprese le matrici, la manipolazione di stringhe di caratteri, di giochi animati e dell'uso di cassette di nastro come memoria di supporto. Questo è il momento giusto per pensare ai metodi di studio e revisionarli, se il caso. Ricordarsi le regole d'oro:

1) Leggere ogni unità completamente, dall'inizio alla fine prima di iniziare a studiarla in dettaglio.
2) Svolgere *tutti* i problemi pratici che sono stati scelti per illustrare i punti essenziali del BASIC.
3) Non passare alla successiva Unità fino a che non si è completamente sicuri della prima. Se per caso ci si blocca, cercare di ritornare indietro di un paio di Unità e ripetere il lavoro.
4) Non correre. Le ultime Unità nel corso richiederanno quattro o cinque giorni ciascuna per poter essere assorbite a fondo.
   Buona fortuna!

UNITA':16

Le funzioni DATA, READ e RESTORE	PAGINA 157
L'analisi nelle monete	157
Uso di un'iterazione	158
Le istruzioni DATA e READ	158
ESPERIMENTO 16.1	159
Formato delle istruzioni DATA	160
Errore delle istruzioni DATA e READ	160
Il comando RESTORE	160
ESPERIMENTO 16.2	161
ESPERIMENTO 16.3	161

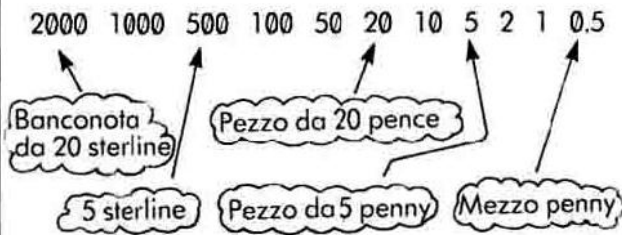
LE FUNZIONI DATA, READ E RESTORE

Questa Unità introduce un importante e utile gruppo di comandi che usano le parole chiave DATA, READ e RESTORE.
I programmi spesso devono fare riferimento a liste di parole o di frasi o serie di numeri che non seguono qualsiasi profilo aritmetico fisso. Per esempio, un programma che calcola e visualizza il calendario per un dato anno deve conoscere i nomi dei giorni della settimana e dei mesi dell'anno. Esso ha inoltre bisogno di una sequenza del numero di giorni di ciascun mese e cioè:

31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

ANALISI DELLE MONETE

Un'altra sequenza comune è la serie di valori delle monete e delle banconote, ad esempio in valuta britannica. Per evitare i decimali, che possono dar luogo a problemi, ci si limiterà al pence:

2000 1000 500 100 50 20 10 5 2 1 0.5

Le persone incaricate di pagare i salari nelle grandi organizzazioni, devono pianificare le operazioni accuratamente ogni settimana. Uno speciale problema sta nell'assicurarsi che ci sia la corretta quantità di cambio per comporre ciascuna busta paga esattamente, usando il minor numero di banconote e di monete.
Si tratta di un utile processo detto "analisi delle monete" che parte da una somma di denaro e la suddivide nel più piccolo numero possibile di banconote e di monete. Per esempio:

158,37 = 7 da 2000P (e cioè 7 pezzi da 20 sterline = 2000 pence)
1 da 1000P
1 da 500P
3 da 100P
0 da 50P
1 da 20P
1 da 10P
1 da 5P
1 da 2P
0 da 1P
0 da 0.5P (e cioè 1/2 penny)

Questo processo, usato su tutti i diversi salari da pagare, aiuta i cassieri addetti a decidere quante banconote e quante monete di ciascun tipo richiedere alla banca.
Si studierà ora un programma per questo tipo di analisi, che potrebbe risultare tanto semplice da non richiedere uno schema di flusso.
Si inizia chiedendo all'utente di fornire una cifra per il salario da suddividere. La cifra iniziale, in pence è memorizzata in W:

```
INPUT "PAGA IN PENCE"; W
```

Successivamente si estrae il numero di banconote da 20 sterline necessarie e lo si inserisce nella variabile. Il numero corretto è il risultato quando W è diviso per 2000, ignorando qualsiasi resto o parte frazionaria. Un comando appropriato è:

```
T = INT(W/2000)
```

Notare che a meno che W non sia una somma di denaro che può essere pagata esattamente in banconote da 20 sterline (ad esempio 80 st.), W/2000 sarà un numero con una parte decimale, ad esempio 7.9185, INT e le parentesi assicurano che venga usata soltanto la parte numerica intera di questa espressione (ad esempio 7) e il resto venga scartato.
Successivamente si visualizzerà il numero di banconote da 20 sterline:

```
PRINT T; "DA 2000P"
```

È così contata una parte sostanziale della busta paga dato che la maggior parte di essa in effetti potrebbe essere pagata interamente in banconote da 20 st. ciascuna. Per continuare il processo di analisi, dobbiamo togliere questo importo dall'importo totale, lasciando solo il resto, che deve essere meno di 20 sterline ossia 2000 pence. L'importo è 2000★T, cosicchè occorre inserire:

```
W = W - 2000★T
```

(nell'esempio W sarebbe ora 15837 − 2000★7 = 1837).
Per la fase successiva, si calcola e si visualizza il numero di banconote da 10 sterline e si diminuisce W di conseguenza. È possibile usare di nuovo la variabile T dato che il suo valore originale (n. di banconote da 20 st.) non è più interessante:

```
T = INT(W/1000)
PRINT T; "DA 1000P"
W = W - 1000★T
```

Le fasi seguenti fino al mezzo penny finale, sono tutte molto simili. Si ottiene:

```
10 INPUT "SALARI IN PENCE"; W
20 T = INT(W/2000)                da 20 st.
30 PRINT T; "DA 2000P."
40 W = W -2000★T
50 T = INT(W/1000)
60 PRINT T; "DA 1000P."           da 10 st.
70 W = W - 1000★T
...
...
140 T = INT(W/50)
150 PRINT T;"DA 50P."             da 50 penny
160 W = W -50★T
...
...
320 T = INT(W/0.5)
330 PRINT T;"DA 0.5P."            da 1/2 penny
340 W = W - 0.5★T
350 STOP
```

Questo programma appare estremamente ripetitivo. Se si potessero accogliere in qualche modo i successivi valori delle banconote e delle monete in una variabile — ad esempio V, l'intero programma di 35 righe, potrebbe essere condensato in una singola iterazione con tre comandi:

```
T = INT(W/V)
PRINT T; "DA;" V; "P."
W = W − V★T
```

Questa iterazione verrebbe eseguita 11 volte; una volta per V=2000, una per V=1000 e così via fino a V=0.5.

USO DI UNA ITERAZIONE

Sarebbe comodo e bello usare un comando FOR, ma ciò non risolverebbe il problema in quanto i valori di V non seguono un profilo determinato. Occorre trovare un altro modo per inserire i valori delle banconote e delle monete in V, quantunque si possa sempre usare un FOR per eseguire l'iterazione 11 volte.
Si consideri una soluzione "stupida". Si sa che un modo per ottenere i numeri in un programma è di fare in modo che l'utente li batta sulla tastiera. Si potrebbe sempre inserire un comando INPUT V nell'iterazione e costringere l'utente a fornire la sequenza di numeri 2000 1000 500 100 . . . 0.5. Il programma apparirebbe pertanto così:

```
10 INPUT "PAGA IN PENCE"; W
20 FOR J=1 TO 11
30 INPUT "SUCCESSIVO VALORE"; V
40 T=INT(W/V)
50 PRINT T; "DA"; V; "P."
60 W=W−T★V
70 NEXT J
80 STOP
```

Impostare questo programma e provare ad eseguirlo. Si otterrà una visualizzazione come questa:

```
PAGA IN PENCE? 9472
SUCCESSIVO VALORE? 2000
4 DA 2000 P.
SUCCESSIVO VALORE? 1000
1 DA 1000 P.
SUCCESSIVO VALORE? 500
0 DA 500 P.
. . .
```

LE ISTRUZIONI DATA E READ

Naturalmente ci sono molti motivi che rendono questo programma poco pratico... È un lavoro molto duro per l'utente e se si verifica un singolo errore nella battitura delle sequenze 2000, 1000, 500, 1000... l'intera analisi viene distrutta e deve essere ripresa da capo. I progettisti del BASIC hanno superato questo problema in un modo ingegnoso ed elegante. Si supponga che il **64** possa battere propri numeri man mano che procede. Si potrebbe usare una tastiera "fantasma" in modo che i numeri non compaiano sullo schermo, e la battitura verrebbe fatta istantaneamente consentendo alla macchina di operare alla sua massima velocità. Cosa dire dei numeri da battere sulla tastiera fantasma? Questi verrebbero immessi in anticipo, sotto forma di istruzioni DATA.
Cambiare ora la riga 30 del programma nel **64** e aggiungere alcune righe DATA per ottenere

```
  10 INPUT "PAGA IN PENCE"; W
  20 FOR J= 1 TO 11
  30 READ V                 cambiata questa riga
  40 T=INT(W/V)
  50 PRINT T; "DA"; V; "P."
  60 W=W−V★T
  70 NEXT J
  80 STOP
1000 DATA 2000              da qui nuove righe
1010 DATA 1000
1020 DATA 500
1030 DATA 100
1040 DATA 50
1050 DATA 20
1060 DATA 10
1070 DATA 5
1080 DATA 2
1090 DATA 1
1100 DATA 0.5
```

Se si esegue questa nuova versione del programma, si ha un'analisi del denaro per qualsiasi cifra immessa. Occorre soltanto indicare la cifra da analizzare. Il programma ha 8 righe di codice e quindi un'istruzione DATA per ciascun valore della banconota o della moneta. Questo è chiaramente un miglioramento rispetto alla versione originale che aveva 35 comandi.
Esaminiamo ora il programma ulteriormente. Il comando READ nella riga 30 è molto simile a INPUT. La parola chiave READ può essere seguita dai nomi di una o più delle variabili, che possono essere numeri o stringhe. La differenza importante è che il comando ottiene le sue informazioni da un'istruzione DATA incorporata nel programma anzichè dall'utente. Se lo si desidera, è possibile immaginare un diavoletto che vive all'interno del **64** che ha una propria tastiera privata. Ogni volta che il **64** obbedisce ad un comando READ, il demonietto trova un'istruzione DATA e rapidamente batte i suoi contenuti su questa tastiera. Egli ricorda quali istruzioni ha usato e le elabora nell'ordine dei rispettivi numeri di label. Pertanto, quando la macchina esegue il comando READ (alla 30) per la prima volta, il diavoletto trova la prima istruzione DATA e batte il numero che essa contiene: 2000. Questo valore viene assegnato a V. La seconda volta il valore usato è 1000 e così via.
Vediamo ora come il comando READ può manipolare stringhe nonchè numeri. Si supponga di voler che il programma di analisi del denaro usi nomi descrittivi per le banconote e le monete ad esempio

1 PEZZO da 5 PENNY

invece dei simboli

1 DA 5 P

I nomi che ci servono possono essere inclusi nelle istruzioni DATA unitamente ai valori stessi. Il comando READ imposterà ora due variabili: il valore V e il corrispondente nome N$. Il programma modificato con le modifiche nelle righe 30, 50 e le istruzioni DATA è:

```
  10 INPUT "SALARI IN PENCE"; W
  20 FOR J = 1 TO 11
  30 READ V, N$
  40 T = INT(W/V)
  50 PRINT T; N$
  60 W = W-V*T
  70 NEXT J
  80 STOP
1000 DATA 2000, BANCONOTA/E DA
     20 STERLINE
1010 DATA 1000, BANCONOTA/E DA
     10 STERLINE
1020 DATA  500, BANCONOTA/E DA
     5 STERLINE
1030 DATA  100, BANCONOTA/E DA
     1 STERLINA
1040 DATA   50, PEZZO/I DA 50 PENNY
1050 DATA   20, PEZZO/I DA 20 PENNY
1060 DATA   10, PEZZO/I DA 10 PENNY
1070 DATA    5, PEZZO/I DA  5 PENNY
1080 DATA    2, PEZZO/I DA  2 PENCE
1090 DATA    1, PEZZO/I DA  1 PENNY
1100 DATA  0.5, PEZZO/I DA ½ PENNY
```


ESPERIMENTO 16·1

Modificare il programma dell'analisi del denaro in modo che funzioni per il sistema monetario degli Stati Uniti d'America. I valori e i rispettivi nomi sono:

Banconota/e da 50$
Banconota/e da 10$
Banconota/e da 5$
Dollaro/i
Quarto/i di dollaro $ 0.25
DIME 1/10 di dollaro $ 0.10
NICKEL 5 cents $ 0.05
PENNY centesimo $ 0.01

Esperimento 16.1 completato	

FORMATO DELLE ISTRUZIONI DATA

Ci sono poche e semplici regole che occorre conoscere a proposito delle istruzioni DATA.
Innanzitutto le istruzioni DATA possono contenere stringhe e numeri misti in qualsiasi ordine. La lunghezza massima di un'istruzione è di 2 righe di schermo (80 caratteri). Se un'istruzione DATA comprende più di un elemento, gli elementi sono separati da virgole. Non occorre inserire virgolette davanti e dietro a una stringa a meno che la stringa non comprenda una virgola o un carattere di controllo dello schermo ad esempio SHIFT e CLR/HOME. Per esempio, l'istruzione DATA

DATA 21, QUEEN ST.

contiene due elementi (un numero e una stringa) mentre

DATA "21, QUEEN ST"

contiene soltanto un elemento: la stringa 21, QUEEN ST.
Secondo, il numero di elementi in ciascuna istruzione DATA non deve corrispondere al numero di variabili nel comando READ. Ciascun READ prende tanti elementi quanti ne occorrono, e se ciò usa solo una parte di riga, non ha importanza; il successivo READ inizia da dove è terminato il primo. Analogamente un'istruzione DATA che contiene troppi elementi, farà semplicemente sì che il **64** vada alla successiva istruzione DATA non appena è necessario.
Per illustrare questo punto, il secondo programma di analisi del denaro funzionerà ancora correttamente se i dati sono ridisposti come segue:

```
1000 DATA 2000
1010 DATA BANCONOTE DA 20 STERLINE
1030 DATA 1000
1040 DATA BANCONOTE DA 10 STERLINE
```

oppure

```
1000 DATA 2000, BANCONOTE DA 20
STERLINE, 1000, BANCONOTE DA 10
STERLINE, 500, BANCONOTE DA 5 STERLINE,
100,...
```

o ancora

```
1000 DATA 2000
1010 DATA BANCONOTE DA 20 STERLINE, 1000
1020 DATA BANCONOTE DA 10 STERLINE, 500,
BANCONOTE DA 5 STERLINE, 100
1030 DATA BANCONOTE DA 1 STERLINA
```

In altre parole, il diavoletto **64** tratta rapidamente con i contenuti delle istruzioni DATA sulla tastiera senza essere troppo preoccupato di dove l'istruzione termina o dove inizia la successiva.
Terzo, le istruzioni DATA non fanno parte del programma allo stesso modo dei vari comandi. Ogniqualvolta si batte RUN, le istruzioni DATA vengono effettivamente selezionate e inserite in una pila diversa prima che il primo comando venga eseguito. Ciò significa che quando un programma viene impostato, le istruzioni DATA possono essere davanti ad esso, dopo di esso o tra i comandi. Per esempio, il programma dell'analisi del denaro potrebbe essere stato scritto con un'istruzione DATA a righe alterne. In ogni caso, questo sarebbe stato un esempio di cattiva prassi di programmazione. Ammucchiare il programma in questa maniera ne avrebbe reso difficile la lettura e non è cosa consigliata. Un'utile convenzione consiste nel porre tutte le istruzioni DATA insieme al termine o all'inizio del programma, e nel dare ad esse numeri di label che sono immediatamente riconoscibili.

ERRORI DELLE ISTRUZIONI DATA E READ

Possono verificarsi due tipi di errori con le istruzioni DATA e comandi READ. Se si dà un comando READ quando tutte le istruzioni DATA sono già state usate (o se non ce n'è), si ottiene un errore OUT OF DATA. Ciò spiega cosa succede se si batte RETURN quando il cursore è in una riga con READY.
Il **64** pensa che si estenda READ Y. Se READ specifica una variabile numerica e la successiva voce nell'istruzione DATA non è un numero, si ottiene un errore di sintassi nell'istruzione DATA (non nel comando READ). Ad es. se si immette:

```
10 READ A
...
...
100 DATA HELLO
```

si ottiene

```
? SYNTAX
ERROR IN 100
```

Ciò può essere motivo di confusione dato che l'errore reale è più probabilmente nel comando READ; si voleva probabilmente inserire:

```
10 READ A$
```

Vale la pena di notare che non c'è corrispondente difetto nell'altro modo: se si legge un numero in una variabile stringa questo viene considerato come stringa di cifre senza segnalare un errore. Perchè no? Una stringa può essere qualsiasi sequenza di caratteri, compresa una sequenza di cifre.

IL COMANDO RESTORE

Infine citeremo il comando RESTORE. Questo comando prende il **64** e lo riporta all'inizio del mucchio d'istruzioni DATA cosicchè queste possono essere lette da capo. RESTORE può essere usato in qualsiasi momento anche se le istruzioni DATA non sono state utilizzate per nulla.

ESPERIMENTO 16·2

a) Scrivere un programma per visualizzare i mesi dell'anno, uno per riga. Le ultime righe del programma dovrebbero essere:

```
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1030 DATA SETTEMBRE, OTTOBRE,
     NOVEMBRE, DICEMBRE
```

b) Scrivere un programma per leggere una data (nella forma giorno-mese-anno) e visualizzare la data e l'anno in cifre, ma il mese in lettere. Per esempio un input di

22,6,1936

dovrebbe dare 22 GIUGNO 1936

Usare le stesse istruzioni DATA come nel precedente programma. Scrivere il programma nella forma di un'iterazione includendo RESTORE in modo che quando viene visualizzata una data, ne viene immessa un'altra.

Esperimento 16.2 completato	

ESPERIMENTO 16·3

L'istruzione DATA è preziosissima nei programmi che presentano quiz o domande. Studiare il seguente programma, immetterlo sul **64** e provarlo.

```
  10 READ A$
  20 IF A$ = "FINE" THEN 190
  30 READ B$
  40 PRINT " SHIFT e CLR HOME "
  50 PRINT A$
  60 PRINT
  70 INPUT X$
  80 PRINT
  90 IF X$ = B$ THEN 130
 100 PRINT "SBAGLIATO. LA RISPOSTA È"
 110 PRINT B$
 120 GOTO 140
 130 PRINT "ESATTO"
 140 PRINT
 150 PRINT "PREMI QUALSIASI TASTO"
 160 GET C$
 170 IF C$ = " " THEN 160
 180 GOTO 10
 190 STOP
 500 DATA CAPITALE DELLA FRANCIA,
     PARIGI
 510 DATA TOKYO E CAPITALE
     DEL, GIAPPONE
 520 DATA PAESE CON MAGG. NUM.
     ABITANTI, CINA
 530 DATA PAESE A SUD DEGLI USA,
     MESSICO
1000 DATA END
```

Glossario:
A$: Testo della domanda
B$: Risposta corretta
X$: Risposta dell'allievo
C$: Usato in attesa di "qualsiasi tasto"

Read A$
(Domanda)

A$="END"
?

Si

Stop

No

Read B$
(Risposta vera)

Visualizza
la domanda

Immissione della
risposta
dell'allievo X$

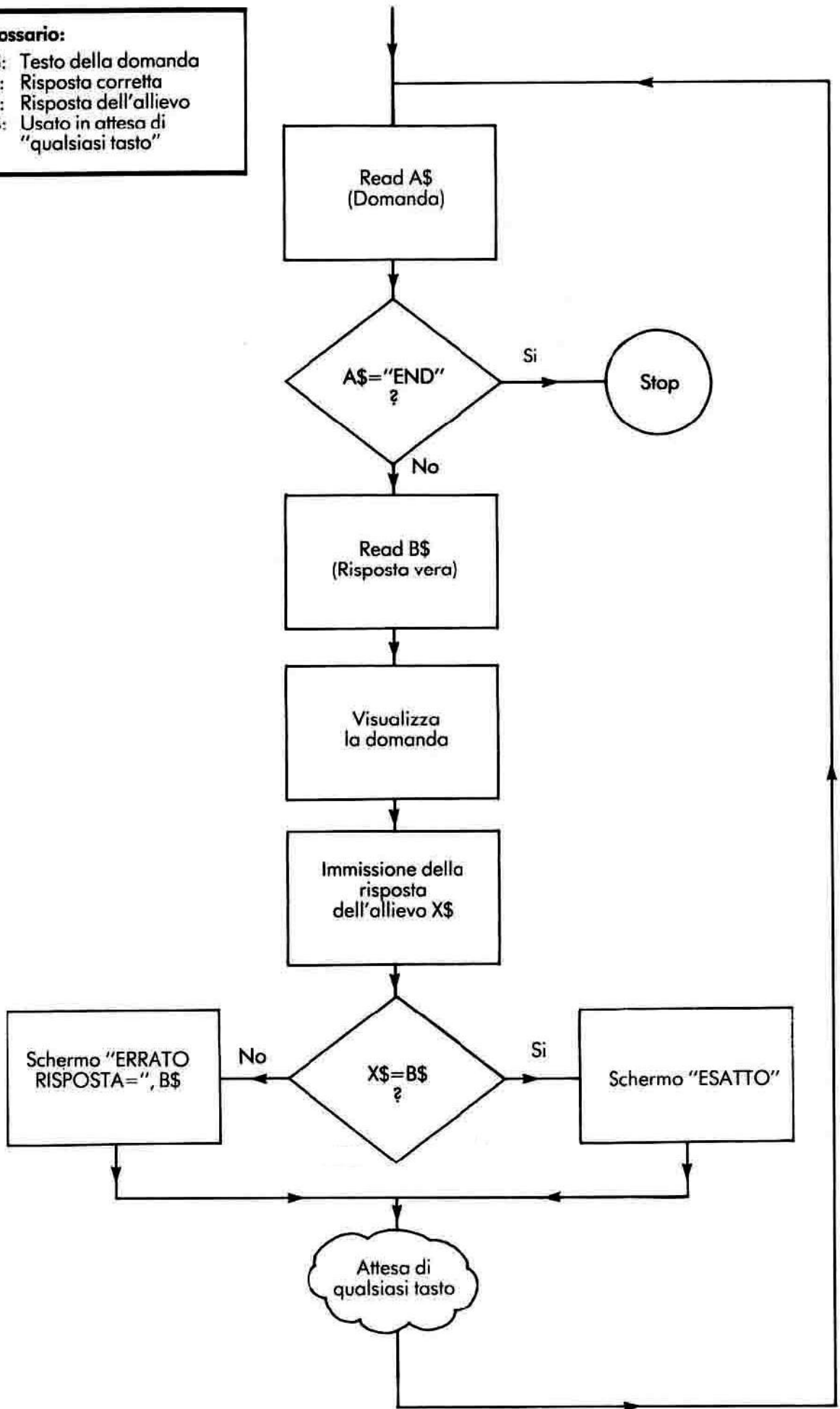
X$=B$
?

No

Schermo "ERRATO
RISPOSTA=", B$

Si

Schermo "ESATTO"

Attesa di
qualsiasi tasto

Una volta fatto girare il programma, creare alcune domande e aggiungerle usando i numeri di label da 540 in poi.
Questo programma quiz di base può essere migliorato in vari mod: per esempio, si potrebbe permettere che l'allievo faccia due o tre tentativi prima di visualizzare la risposta esatta e si potrebbe contare il numero di risposte corrette e visualizzare una percentuale al termine della lezione.
Scrivere un programma di quiz migliorato per porre domande su un argomento preferito.

Esperimento 16.3 completato	

Il quiz di auto-test per questa Unità è detto UNIT16QUIZ

UNITA':17

Come affrontare la complessità dei programmi	165
Uso del due punti	166
Altri modi per usare le istruzioni IF-THEN	166
ESPERIMENTO 17.1	167
Operatori logici	169
L'operatore AND	169
L'operatore OR	170
Combinazioni di operatori logici	170
Il comando NOT	171
ESPERIMENTO 17.2	171

COME AFFRONTARE LA COMPLESSITÀ DEI PROGRAMMI

Ora incontreremo il più grande problema del programmatore — il controllo della complessità. Molte persone comprendono i principi della programmazione e possono scrivere brevi e semplici programmi con facilità, ma quando applicano gli stessi metodi ai problemi più complessi ottengono scarso successo. Questo sembra essere un limite alla quantità di dettagli che è possibile tenere in mente in qualsiasi momento. Quando questo limite viene superato, il risultato è confusione, quantità di gravi errori e programmi che sistematicamente danno risposte sbagliate. Nessuno dei sussidi descritti nella Parte 1 (ad esempio il controllo nell'Unità 8 e lo schema di flusso nell'Unità 11), danno molto aiuto se usati isolatamente. Tutto è ancor troppo complicato.

In questa unità si considereranno alcuni dei modi per ridurre la complessità di un programma, senza deviare dal lavoro che si suppone esso debba fare. Chi ambisce diventare un programmatore, deve studiare questi metodi e usarli sistematicamente in tutto il lavoro. Essi saranno di aiuto dando soluzioni invece che bloccare su un problema, cercando di risolverlo con modifiche casuali, per vedere se per fortuna è possibile imbattersi in quella che sembra funzionare.

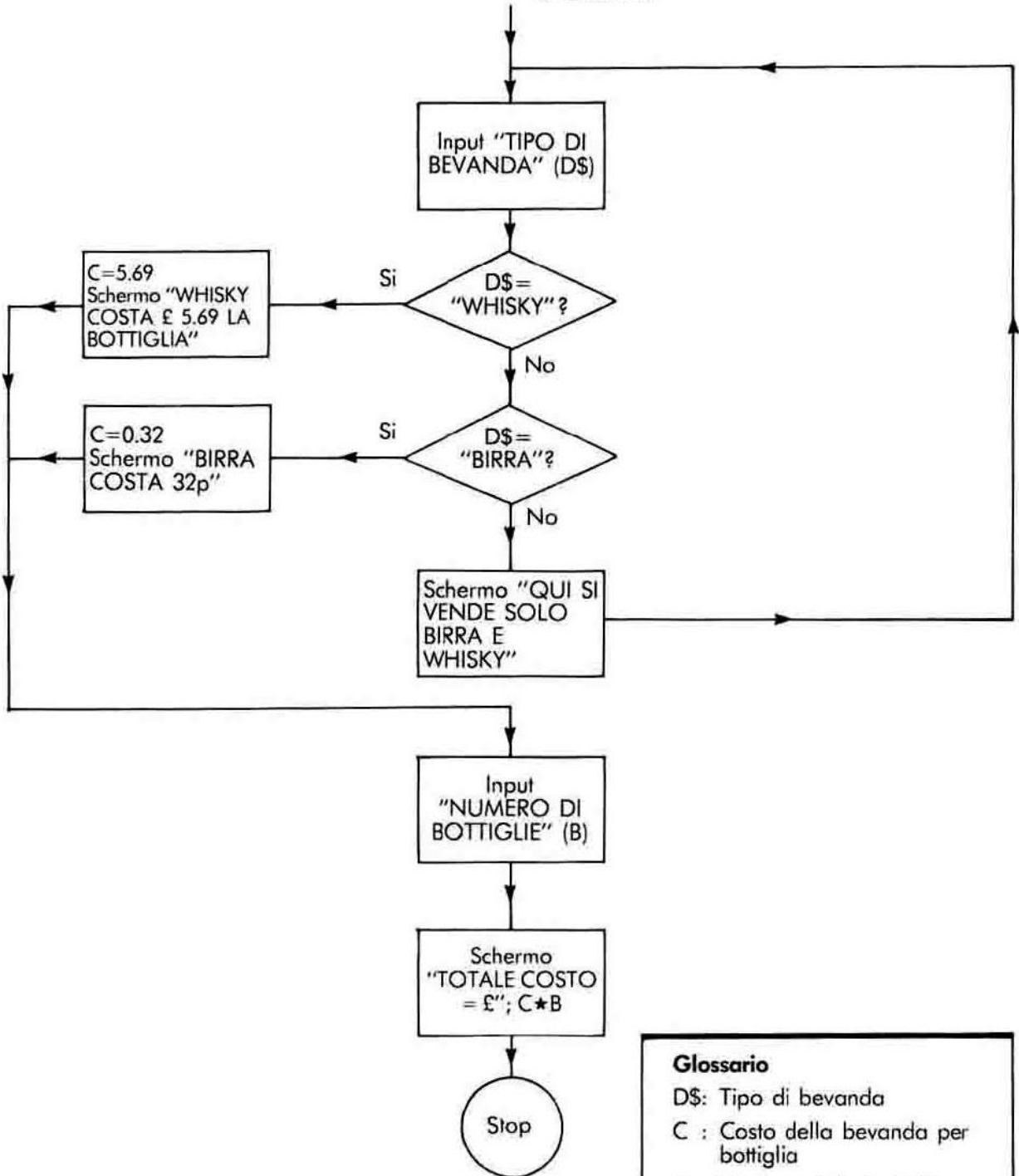
Uno dei principali vantaggi degli schemi di flusso nel disegno dei programmi è che essi indicano la struttura di un programma molto meglio che non segmenti di codice BASIC.

Si confronti:

Glossario

D$: Tipo di bevanda

C : Costo della bevanda per bottiglia

B : Numero delle bottiglie

con:

```
 10 INPUT "TIPO DI BEVANDA"; D$
 20 IF D$ = "WHISKY" THEN 70
 30 IF D$ = "BIRRA" THEN 100
 40 PRINT "SI VENDE SOLO BIRRA"
 50 PRINT "E WHISKY"
 60 GOTO 10
 70 C = 5.69
 80 PRINT "WHISKY COSTA 5.69"
 90 GOTO 120
100 C = 0.32
110 PRINT "BIRRA COSTA 32P."
120 INPUT "QUANTE BOTTIGLIE"; B
130 PRINT "TOTALE COSTO = £"; C*B
140 STOP
```

L'effetto di tradurre lo schema di flusso in BASIC è ovvio; una chiara serie di istruzioni è stata spiacciccata in una lista unidimensionale senza forma di comandi che devono essere disaggrovigliati prima di avere un senso.

USO DEL DUE PUNTI

Il Microsoft BASIC, la versione del linguaggio usato sul **64** , ha alcune utili caratteristiche che gli consentono di conservare la maggior parte della struttura di uno schema di flusso.
Un'intera sequenza di comandi può essere raggruppata riunendo i comandi stessi dopo lo stesso numero di label e separandoli con il simbolo del due punti ":" senza un RETURN. L'effetto è lo stesso che se il comando fosse stato scritto su righe separate salvo che è impossibile saltare uno dei comandi intermedi mediante un GOTO o un IF. Per esempio, la sequenza

```
10 INPUT "COME TI CHIAMI"; N$
20 PRINT "HELLO"; N$
30 PRINT
40 S = 0
50 Q = 100
```

potrebbe essere sostituita da

```
10 INPUT"COME TI CHIAMI";N$:
   PRINT"HELLO";N$:PRINT:S=0:Q=100
```

posto che nessun'altra parte del programma debba fare un salto ad uno qualsiasi dei comandi originariamente numerati da 20 a 50. Il numero limite dei comandi che possono essere raggruppati è definito dalla massima lunghezza per istruzione (80 caratteri, 2 righe di schermo).

ALTRI MODI DI USARE LE ISTRUZIONI IF-THEN

Il comando THEN in un IF-THEN non deve sempre essere seguito da un numero di label ma può anche essere seguito da un comando (o da un gruppo di comandi) che vengono eseguiti soltanto se la condizione è vera. Ciò significa che è possibile sostituire

```
10 IF X=0 THEN 30
20 GOTO 40
30 PRINT "X=0"
40 ———
```

con

```
10 IF X=0 THEN PRINT "X=0"
20 ———
```

A questo punto è necessaria un po' di attenzione. Se la condizione in un comando IF-THEN è falsa, il **64** trasferisce sempre il controllo all'istruzione con label successiva. Ne segue che se un comando IF-THEN fa parte di un gruppo di comandi separati da due punti qualsiasi comando che lo segue sarà inevitabilmente saltato se la condizione è falsa. Ciò potrebbe non essere quello che si intende fare! Per illustrare questo punto si consideri la sequenza

```
10 IF X=0 THEN 20: Y=5: GOTO 30
20 Y=7
30 PRINT Y
```

Osservando il programma ci si può domandare cosa significa: il programmatore voleva che Y venisse definito a 7 se X era 0 o a 5 se non era 0. Sfortunatamente ciò non è quello che effettivamente succede. Si consideri il comando sulla riga 10:

```
IF X=THEN 20
```

Se la condizione è vera (e cioè X=0), il **64** salta al comando 20, esattamente come ci si aspetterebbe. Se la condizione è falsa, la macchina segue la regola e trasferisce il controllo all'istruzione con label successiva che risulta essere 20! I comandi

```
Y=5: GOTO 30
```

non verranno mai eseguiti in nessun caso. Questa trappola viene evitata seguendo una regola semplice: se un comando IF-THEN coinvolge un salto ad una label, questo deve essere seguito da un RETURN — non da un due punti.
Usando questa nuova funzione, il programma del prezzo delle bevande precedentemente discusso può essere abbreviato e chiarito:

```
10 INPUT "TIPO DI BEVANDA"; D$
20 IF D$="WHISKY"THENC=5.69:PRINT
   "WHISKY COSTA £ 5.69.":GOTO 50
30 IF D$ ="BIRRA"THEN C=0.32: PRINT
   "BIRRA COSTA 32P":GOTO 50
40 PRINT "VENDITA SOLO BIRRA E":PRINT
   "WHISKY":GOTO 10
50 INPUT "QUANTE BOTTIGLIE";B
60 PRINT "TOTALE COSTO= £";C*B
70 STOP
```

Confrontare le due versioni e notare che la seconda si conforma molto più da vicino alla struttura dello schema di flusso originale.

Riscrivere i seguenti programmi usando il minor numero possibile di comandi con label:

a)
```
10 INPUT "QUANTI MINUTI"; M
20 R = TI+ M★3600
30 IF TI <R THEN 30
40 PRINT "TEMPO SCADUTO"
50 STOP
```

b)
```
 10 PRINT "USA 1000000 PER
 20 PRINT "TERMINARE INPUT"
 30 S=0
 40 N=0
 50 INPUT "NUMERO SUCCESSIVO"; X
 60 IF X = 1000000 THEN 100
 70 S=S+X
 80 N=N+1
 90 GOTO 50
100 PRINT "MEDIA="; S/N
110 STOP
```

c) Caricare il programma intitolato UNIT 17 PROG e listarlo. Si vedrà che si suppone che esso riconosca i nostri JIM, BOB, KATE e PENNY e dica di che cosa mancano. Sfortunatamente il programma non funziona. Correggerlo.

Esperimento 17.1 completato	

OPERATORI LOGICI

Ulteriore aiuto nella semplificazione dei programmi è dato dagli operatori logici AND, OR e NOT. Queste parole hanno significati speciali in BASIC e sono usate per concatenare semplici condizioni nei comandi IF-THEN cosicchè si possono prendere decisioni più complesse.

L'OPERATORE "AND"

Iniziamo con AND, l'operatore logico più frequentemente usato. Esso generalmente si trova tra due condizioni come in questo caso:

IF A> 18 AND M$ = "Q" THEN . . .

La risultante condizione composta (che è tutto ciò che si trova tra IF e THEN) è *vera* soltanto se entrambe le condizioni semplici sono a loro volta *vere*. Se una o l'altra (o entrambe) sono *false*, la condizione composta è *falsa*.
L'operatore AND generalmente consente di sostituire due o più comandi IF-THEN con uno solo. Si consideri un programma che esamini le domande per entrare in una società di polizia privata. Le regole dicono che i candidati devono avere almeno 18 anni di età ed essere alti almeno 160 cm. Lo schema di flusso probabilmente comprenderà una sezione di questo tipo:

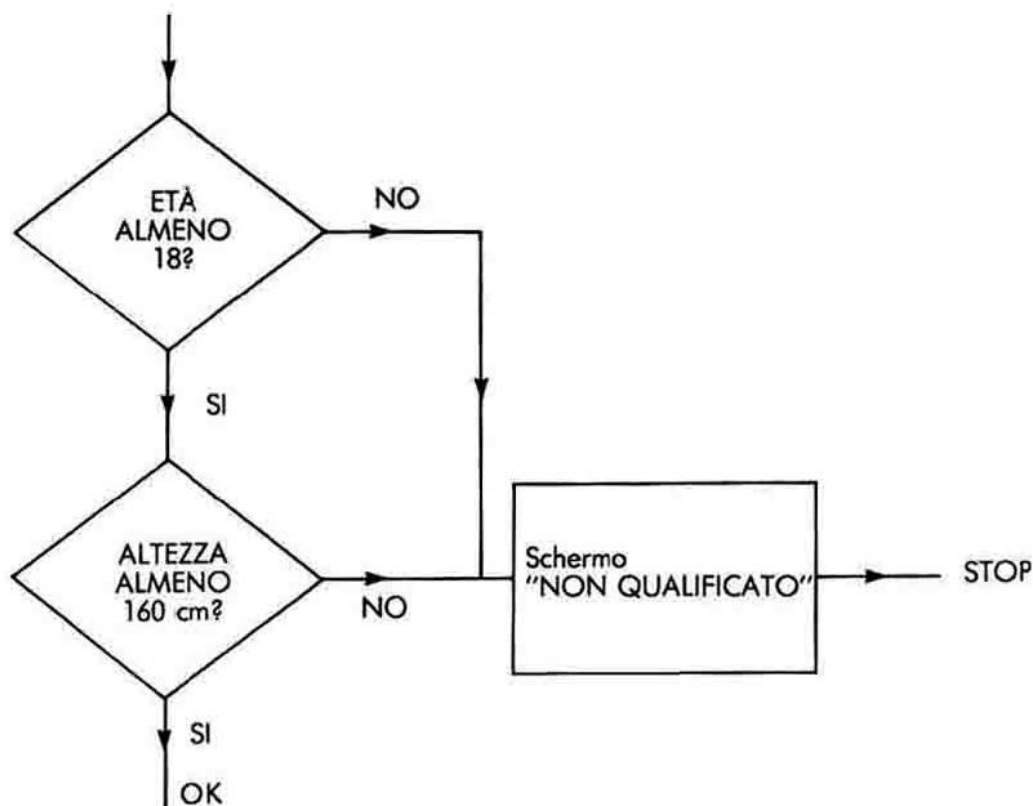



Figure 17.2

Se fossero ammesse solo condizioni semplici, la sequenza potrebbe essere codificata nella forma:

```
100 IF A > = 18 THEN 130
110 PRINT "NON QUALIFICATO"
120 STOP
130 IF H < 160 THEN 110
140 REM ETÀ E ALTEZZA ACCETTABILI
150...
```

Usando una condizione composta, che applica le prove per l'età e l'altezza contemporaneamente, la sezione può essere scritta in maniera molto più compatta e con maggior eleganza. Il suo significato è immediatamente chiaro:

```
100 IF A > = 18 AND H > = 160 THEN
120
110 PRINT "NON QUALIFICATO": STOP
120 REM ETA E ALTEZZA ACCETTABILI
```

AND è un operatore piuttosto simile a ★, salvo che usa condizioni anzichè numeri. Ciò significa che le condizioni possono essere concatenate indefinitamente, fino al limite della lunghezza interna di 80 caratteri. Si potrebbe inserire:

```
IF A=5 AND B<7 AND X$<>"GIUSEPPE"
AND... THEN 1260
```

La risultante istruzione composta è vera soltanto se *tutte* le condizioni semplici sono a loro volta vere.
Un uso speciale di AND è di decidere se una variabile cade in un campo specifico. È possibile provare se il valore della variabile X cade ad esempio tra 7 e 12 compresi. Un matematico esprimerebbe quest'idea scrivendo:

7< = X< = 12

ma non è possibile inserire tale "condizione" in un comando IF-THEN — non è in linguaggio BASIC. Per contro, si usano due semplici condizioni concatenate con AND:

```
IF X>=7 AND X< = 12 THEN...
```

L'OPERATORE "OR"

L'operatore OR è usato più o meno allo stesso modo di AND, ma produce una condizione vera se una o l'altra o entrambe le due condizioni coinvolte sono vere. Un uso comune di OR è per controllare che la risposta ad una domanda sia una di quelle previste. Per esempio:

```
10 PRINT"SEI IN GAMBA? RISPONDI"
20 INPUT "SI O NO"; A$
30 IF A$="SI" OR A$="NO" THEN 50
40 PRINT "RISPONDI ALLA DOMANDA!";:
   GOTO 10
50 REM RICEVUTA RISPOSTA VALIDA
```

La condizione composta può essere ampliata per includere parecchi OR come in:

```
IF H$="NERO"OR H$="MARRONE"OR
H$="ROSSO"ORH$="ONESTO"THEN
30
```

Notare che una forma che non è possibile usare (dato che non è in linguaggio BASIC corretto), è:

```
IF H$="NERO" OR"MARRONE"OR"ROSSO"
OR "ONESTO" THEN 30
```

COMBINAZIONI DI OPERATORI LOGICI

È spesso comodo usare condizioni composte che sfruttano più di un tipo di operatore logico. Per esempio, le regole per emettere le patenti di guida, possono dire che il richiedente deve avere un'età superiore ai 16 anni per guidare un motociclo, superiore ai 18 anni per guidare un'automobile e superiore ai 21 anni per guidare un autobus. È possibile porre:

```
IF V$="MOTOCICLO" AND A >=16
OR V$="AUTO" AND A >=18
OR V$="BUS" AND A>=21
THEN PRINT "OK"
```

Quando il **64** viene ad elaborare questa condizione composta, lo fa in un ordine particolare; prima di tutto le condizioni semplici, quindi gli AND e infine gli OR. In questo esempio, il processo dà esattamente il risultato richiesto.
In pratica, le condizioni composte potrebbero non sempre essere così facili da scrivere. Si consideri l'esempio di un'azienda che cerca un programmatore con tre anni di esperienza nel linguaggio di programmazione BASIC o COBOL. Se si ponesse:

```
IF E>=3 AND L$="BASIC" OR
            L$="COBOL"
```

(E è il numero di anni di esperienza, L$ il linguaggio)

le regole di valutazione sceglierebbero:

a) Un programmatore BASIC con almeno 3 anni di esperienza

o

b) Un programmatore COBOL con al limite nessuna esperienza.

Ciò in quanto viene usato per primo AND e si associa E>=3 con "BASIC" ma non non "COBOL". Il modo per evitare questo problema consiste nell'usare la parentesi. Esattamente come nell'algebra ordinaria, tutto ciò che si trova all'interno di parentesi viene elaborato prima di qualsiasi cosa che si trovi all'esterno. Scrivendo

```
IF E>=3 AND (L$="BASIC' OR L$=
"COBOL")
```

si ottiene l'ordine corretto e pertanto la risposta sarà corretta.

IL COMANDO "NOT"

NOT è il terzo operatore logico. Esso viene applicato ad una condizione (semplice o composta) e ne inverte il senso. Per esempio, se X>5 è vero, NOT X >5 è falso e viceversa.
Non è mai necessario usare NOT con condizioni semplici dato che può essere usata tranquillamente la relazione "opposta". Pertanto

NOT X=5 è lo stesso che scrivere X<>5
NOT X<>5 è lo stesso che scrivere X=5
NOT X<5 è lo stesso che scrivere X>=5
NOT X>5 è lo stesso che scrivere X<=5
NOT X<=5 è lo stesso che scrivere X>5
NOT X>=5 è lo stesso che scrivere X<5

NOT entra in scena con proprie condizioni composte, come si vedrà più avanti.
La regole del BASIC specificano che in assenza di parentesi, NOT debba essere usato prima di qualsiasi altro operatore logico. Si è visto che è inutile usarlo per invertire condizioni semplici. Per far sì che esso affronti un'intera condizione composta, la condizione deve essere racchiusa tra parentesi come questa:

```
IF NOT (X=5 AND Y=7) THEN...
```

Una condizione composta con una NOT potrebbe essere usata per rilevare e rifiutare talune risposte vietate. Ciò è illustrato dalla sequenza:

```
10 PRINT "COSA PENSI"
20 INPUT "DI QUESTO"; R$
30 IF NOT (R$="FLAGELLO" OR R$=
   "DISGRAZIA" OR R$="BESTEMMIA")
   THEN 50
40 PRINT "BADA A COME PARLI!"
   GOTO 10
50 REM RISPOSTA NON VERA
```

Le condizioni composte precedute da NOT possono essere spesso semplificate. Se ogni operatore logico all'interno delle parentesi è un OR, il NOT e le parentesi possono essere tolti posto che:

a) Ciascuna condizione semplice sia invertita
b) Ogni OR sia cambiato in un AND.
   Pertanto la riga 30 nel nostro esempio, potrebbe essere scritta:

```
30IFR$<>"FLAGELLO"ANDR$<>"DISGRAZIA"
   AND R$<>"BESTEMMIA" THEN 50
```

Una regola simile si applica alle condizioni composte invertite in cui ogni operatore è un AND: gli AND diventano OR, le condizioni semplici sono invertite e il NOT e le parentesi vengono tolti. Queste due regole sono dette "leggi di De Morgan" da chi le ha scoperte. La maggior parte dei manuali di testo e di programmazione, raccomandano di evitare le condizioni NOT ogniqualvolta possibile ed è solitamente più facile procedere in questo modo.

a) Usare le leggi di De Morgan per esprimere le seguenti condizioni composte senza usare il NOT:

```
NOT (N$="BIANCHI" OR N$="ROSSI"
     OR N$="MAURI")
NOT (X< = 15 AND X >=4)
```

Si supponga che un programma abbia misurato un tempo di reazione (in secondi) e lo abbia inserito nella variabile T. Scrivere una sezione di codice che visualizza uno dei seguenti commenti a seconda dei casi:

Valore di T	Commento
T < 0.1	FANTASTICO!!
0.1 < = T <0.15	ECCEZIONALE
0.15 < = T < 0.2	MOLTO BUONO
0.2 < = T < 0.25	BUONO
0.25 < = T < 0.28	DISCRETO
0.28 < = T < 0.33	MOLTO LENTO
0.33 < = T < 0.4	SVEGLIA!
0.4 < T	PROVA ANCORA QUANDO SEI SOBRIO!!

c) C'è un'enciclopedia in quattro volumi:
   1: Da ABRAHAM a FRANCE
   2: Da FRANCHISE a LEVANTE
   3: Da LEVITAZIONE a QUOTA
   4: Da QUOZIENTE a XILOFONO

Scrivere un programma che immetta qualsiasi parola e mi dice in quale volume cercarla. Il programma dovrebbe anche dire se la parola non è inclusa (ad esempio "QUOTO").
Suggerimento: ricordare che gli operatori <, > = e altri del genere, possono essere usati come stringhe e danno risultati secondo il loro ordine alfabetico.

Esperimento 17.2 completato	

UNITA':18

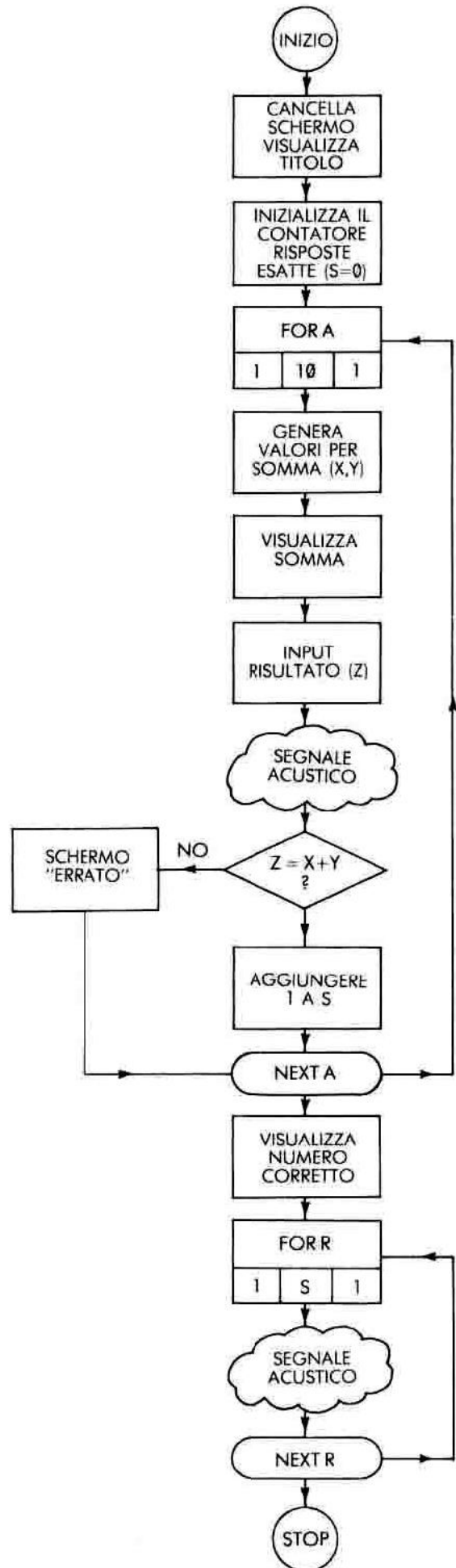
Introduzione alle subroutine	PAGINA 175
Come funziona GOSUB	176
ESPERIMENTO 18.1	177
Subroutine con compiti variabili	178
Trasmissione di parametri alle/dalla subroutine	178
Decisioni nel progettare le subroutine	179
Uso di più di un parametro	179
ESPERIMENTO 18.2	180
Subroutine più complesse	181
ESPERIMENTO 18.3	182

INTRODUZIONE ALLE SUBROUTINE

Fino a questo punto nel corso, si è fatta abbastanza pratica scrivendo piccoli programmi — ad esempio fino a 30 comandi o giù di lì. Presto o tardi si sarà portati a scoprire che i programmi più interessanti devono essere molto più lunghi e che occorrono strumenti e tecniche specializzati per aiutare a costruirli correttamente.

Un sussidio vitale nello scrivere grandi programmi è la possibilità di ricorrere alle subroutine. Nello schema di flusso (Unità 11 della Parte 1), si è già parlato della possibilità di inserire azioni complicate all'interno di "nuvolette" e di rimandarne la codifica ad un tempo successivo. È questo un metodo utile per affrontare le complessità in quanto consente di trascurare una gran parte di dettagli e di concentrarsi sui risultati principali del problema.

Le subroutine sono come le nuvolette. La subroutine si compone di un gruppo di comandi che collaborano per eseguire un compito particolare e ben definito. Quando si disegna un programma, si può pensare a questo compito come ad una singola fase, per quanto complicato possa essere.

Inizieremo con una subroutine semplicissima. Si consideri un programma che ponga dei quiz sulle somme.

Glossario

S: Contatore del numero di risposte corrette
X, Y } Valori da sommare
Z: Somma (risultati)
A: Contatore del numero di domande
R: Contatore dell'iterazione di premio

INIZIO
CANCELLA SCHERMO VISUALIZZA TITOLO
INIZIALIZZA IL CONTATORE RISPOSTE ESATTE (S=0)
FOR A
1 | 10 | 1
GENERA VALORI PER SOMMA (X,Y)
VISUALIZZA SOMMA
INPUT RISULTATO (Z)
SEGNALE ACUSTICO
Z = X+Y ?
NO
SCHERMO "ERRATO"
AGGIUNGERE 1 A S
NEXT A
VISUALIZZA NUMERO CORRETTO
FOR R
1 | S | 1
SEGNALE ACUSTICO
NEXT R
STOP

Il compito che abbiamo scelto da usare come una subroutine è la nuvoletta contrassegnata

"Make Sound"

Il codice per questa azione sarebbe normalmente qualcosa del genere:

```
10 VV = 212*256
20 POKE VV+24,15: POKE VV,0
30 POKE VV+1, 80: POKE VV+5,9
40 POKE VV+4,0: POKE VV+4,33
50 FOR M=1 TO 800: NEXT M
```

Per trasformare queste istruzioni in una subroutine occorre "vestirle" in modo che si inseriscano correttamente nel programma principale. In linguaggio tecnico occorre cioè fornire una *interfaccia*.

Innanzitutto si sceglie una serie di numeri di label elevata per non entrare in conflitto con il programma principale o con qualsiasi altra subroutine. Il numero di label più elevato ammesso è 63999.

Secondo si aggiunge un commento (REM) descrittivo all'inizio della subroutine. Ciò non è assolutamente necessario, ma è un segno di programmazione competente ed è molto comodo dato che molte subroutine possono essere usate in altri programmi.

Terzo, si assegna un nome distintivo alla variabile usata dalla subroutine, ad esempio una doppia lettera. Anche in questo caso non è obbligatorio ma segue un'utile convenzione che verrà spiegata più avanti.

Infine, si terminerà la subroutine con il comando speciale:

RETURN

Questo comando, che viene usato al termine di ogni subroutine, deve essere battuto correttamente e interamente (6 lettere) e seguito come al solito da RETURN. Non confondere il comando RETURN e il tasto RETURN — essi sono totalmente diversi. Usando questa convenzione, le istruzioni per la subroutine potrebbero essere:

```
1000 REM SUBROUTINE PER CREARE
     SUONO PIP
1010 VV=212*256
1020 POKE VV+24,15:POKE VV,0
1030 POKE VV+1,80:POKE VV+5,9
1040 POKE VV+4,0:POKE VV+4,33
1050 FOR MM=1 TO 800:NEXT MM
1060 RETURN
```

Ora può essere scritto il programma principale. Ogniqualvolta si ha una nuvoletta con l'istruzione "CREA SUONO" (segnale acustico), questa può essere tradotta dal comando di richiamo della subroutine:

GOSUB 1000

solitamente seguito da un REM sulla stessa riga per spiegare ciò che viene fatto. L'intero programma (compresa la subroutine) si trasforma come segue:

```
  10 PRINT " SHIFT e CLR HOME
     PROVA ARITMETICA"
  20 PRINT "CALCOLA SOMME
     SEGUENTI"
  30 S=0
  40 FOR A=1 TO 10
  50 X = INT(10*RND(0)+1)
  60 Y = INT(10*RND(0)+1)
  70 PRINT X; "+"; Y; "−";
  80 INPUT Z
  90 GOSUB 1000:REM CREA SUONO
 100 IF Z=X+Y THEN 130
 110 PRINT "ERRATO"
 120 GOTO 150
 130 PRINT "ESATTO"
 140 S=S+1
 150 NEXT A
 155 FOR T=1 TO 500: NEXT T
 160 PRINT "ESATTO";S;"SU 10"
 170 FOR R = 1 TO S
 180 GOSUB 1000:REM CREA SUONO
 190 NEXT R
 200 STOP
1000 REM SUBROUTINE PER CREARE
     SUONO PIP
1010 POKE 212*256
1020 POKE VV+24,15:POKE VV,0
1030 POKE VV+1,80:POKE VV+5,9
1040 POKE VV+4,0:POKE VV+4,33
1050 FOR MM =1 TO 800:NEXT MM
1060 RETURN
```

Battere questo programma e verificare che funzioni come ci si aspetta.

COME FUNZIONA GOSUB

Eseguiamo ora attentamente il programma. Il GOSUB è più o meno come GOTO ma con una differenza importante. Quando il **64** salta alla subroutine, ricorda la label del successivo comando del programma principale e memorizza questa informazione in una parte speciale della memoria denominata catasta operativa. Il RETURN al termine della subroutine assomiglia a sua volta ad un GOTO ma la destinazione è sempre il numero di label memorizzata nella catasta! Ciò fornisce un meccanismo automatico che assicura che, ogniqualvolta il **64** finisce l'esecuzione di una subroutine, ritorni sempre indietro al punto corretto nel programma principale. Il numero di label memorizzato nella catasta, mentre il **64** esegue la subroutine, è detto *concatenamento* o *indirizzo di ritorno*.

Il nostro programma inizia eseguendo i comandi nella riga 10 nel modo solito. Il comando 90 dice GOSUB 1000; così il computer salta a 1000 ma per strada nota la successiva istruzione che in questo caso è un'istruzione REM. La locazione d'indirizzo di questa istruzione REM viene messa nella catasta. Questo è il collegamento o indirizzo di ritorno.

Una volta raggiunta la subroutine, la macchina esegue i comandi 1010-1050 la cui ultima riga è RETURN. Dato che la catasta contiene l'indirizzo dell'istruzione REM nella riga 90, il RETURN rimanda la macchina a questo punto nel programma. Dato che si tratta semplicemente di

un'istruzione REM, la macchina procede alla riga successiva.

Quando sono state eseguite tutte le somme, il programma passa ad un altro richiamo di subroutine, alla riga 18Ø. Salta di nuovo alla riga 1ØØØ ma stavolta l'indirizzo di collegamento inserito nella catasta è per l'istruzione REM che si trova alla riga 18Ø. Quando viene eseguito per la seconda volta RETURN, il controllo ritorna all'istruzione REM nella riga 18Ø e non alla riga 9Ø come in precedenza.

Questo semplice esempio mostra come è possibile separare uno dei lavori che costituiscono un programma e trattarlo a parte. Chiaramente, più complessa è la funzione che si deve separare e maggiormente si semplifica il disegno generale del programma. L'esempio mostra anche come è possibile richiamare una subroutine da più di un punto senza doverla scrivere ogni volta. Ciò può essere vero, ma bisogna fare attenzione a coloro che dicono che l'importanza principale delle subroutine consiste nell'abbreviare i programmi. Ciò è falso e ingannevole. Il punto realmente importante della subroutine è la possibilità di semplificare la struttura dei programmi separando sezioni compresse e considerandole isolatamente. Ciò sarà più ovvio nelle illustrazioni successive.

ESPERIMENTO 18·1

a) Modificare il programma a pagina 176 in modo che il margine diventi nero quando una risposta è sbagliata e diventi porpora quando è corretta. Non dimenticarsi di ripristinare il colore iniziale quando viene visualizzata la somma successiva.

b) Usare ora le stesse subroutine per scrivere un programma completamente diverso. Si immagini un bambino a cui viene insegnato a contare. Per ogni domanda il programma emette una serie di pip (tra 1 e 9). Ci si aspetta che l'allievo conti i pip e che batta i numeri adatti. Per esempio, ...pip-pip-pip... ha la risposta corretta "3" e qualsiasi altra cosa è sbagliata.

Esperimento 18.1 completato	

SUBROUTINE CON COMPITI VARIABILI

L'esperimento 18.1 mostra che le subroutine possono essere abbastanza indipendenti dal programma in cui risiedono. Esse possono essere spostate da programma a programma e possono essere scritte da persone diverse. (Il lettore ha appena usato le subroutine scritte dall'autore di questo manuale nell'ultimo programma). È possibile effettivamente comprare librerie di subroutine per eseguire le varie funzioni e i vari lavori e ciò può risparmiare un tempo notevole nella costruzione di un programma.
Il programma con subroutine è paragonabile ad un ufficio con un capo (il programma principale) e parecchi assistenti (le subroutine). Ciascun assistente è specializzato nel fare solo un lavoro, ad esempio prelevare il documento in cima ad un armadio di archivio o fare il caffè. Il capo ha un telefono e può chiamare un assistente in qualsiasi momento e dirgli di fare il suo lavoro speciale. Quindi (almeno in BASIC), il capo attende fino a che l'assistente lo richiama e dice "fatto".
Le subroutine che abbiamo già esaminato erano molto limitate. Ciascuna di esse poteva svolgere soltanto un lavoro preciso, ad esempio cambiare il colore del margine o eseguire un particolare tipo di rumore. In un ufficio dove gli assistenti sono ugualmente inflessibili su ciò che essi possono fare, il solo comando che il capo può dare è "fallo!". Ciò fa sì che l'assistente incaricato del caffè inizi a preparare una tazza di caffè, che è la sola cosa che egli sa fare. Se il capo ha dei visitatori e vuole cinque tazze di caffè, deve chiamare l'assistente del caffè per cinque volte. Gli assistenti in questo ufficio, sarebbero molto più utili se il capo potesse in qualche modo qualificare il lavoro che ad essi assegna. Per esempio, si risparmierebbe tempo se chi fa il caffè fosse in grado di contare e il capo potesse dirgli quante tazze preparare. Sarebbe anche utile se il capo potesse dire all'archivista quale documento prelevare dall'archivio. Gli assistenti sarebbero sempre limitati ad un lavoro ma potrebbero eseguirlo in un modo più flessibile.
Nello stesso modo generale, una subroutine in un programma diventa molto più utile se lo si può chiedere di fare uno qualsiasi di un'intera famiglia di compiti collegati. Per esempio, potrebbe essere comodo per un programma usare una subroutine che emetta qualsiasi numero di "pip" secondo le istruzioni pervenute dal programma principale.
Quest'idea solleva l'interessante questione della comunicazione. Ci si aspetterebbe che i messaggi che vengono scambiati tra il capo e i suoi nuovi versatili assistenti siano più complicati dato che egli deve ora indicare un numero o il titolo di un documento.
Gli assistenti che hanno l'incarico speciale di trovare l'informazione per il capo, possono trasmettergliela quando gli dicono "fatto". Nell'ufficio tutto ciò è abbastanza semplice in quanto c'è un sistema telefonico, ma cosa succede nei programmi?

TRASMISSIONE DI PARAMETRI ALLE/DALLE SUBROUTINE

Molti linguaggi di programmazione tipo PASCAL o ADA, dispongono di speciali meccanismi per la comunicazione tra il programma principale e le subroutine, ma il BASIC, fa le cose in modo ancor più semplice. Le informazioni sono passate in *variabili* che sono usate in comune tra il programma principale e le sue subroutine. Queste variabili hanno un nome speciale: *parametri*. Qualsiasi variabile si comporta come un parametro, ma noi adotteremo una convenzione speciale: ogni nome di parametro dovrà essere costituito da una lettera seguita dalla cifra 1, seguita dal segno $ se il parametro è una stringa. Ecco alcuni esempi:

A1 X1 C1$ G1

Qui c'è un esempio di una subroutine per visualizzare una riga con qualsiasi numero di ★, come segue:

★★★

o

★★★★★★★★★★★

```
3000 REM VISUALIZZA NUMERO DI ★
     DATI IN X1 SU UNA RIGA
3010 FOR JJ = 1 TO X1
3020 PRINT "★";
3030 NEXT JJ
3040 PRINT
3050 RETURN
```

Si esamini da vicino questa routine. I comandi da 3010 a 3030formano un'iterazione che è ripetuta X1 volte. Ogni volta, viene visualizzato un ★ sulla stessa riga degli ★ precedenti. X1 è il *parametro* o la variabile che dice alla subroutine quanti ★ occorrono. JJ è una variabile locale; cioè è usata all'interno della subroutine, ma il suo valore all'esterno della subroutine non ha alcun interesse. La subroutine non è di alcuna utilità a meno che non venga richiamata. Ciò richiede un paio di comandi: uno per definire X1 a un valore appropriato e uno per eseguire il richiamo effettivo. Per ottenere una riga di 17 ★ , il programma potrebbe comprendere:

```
X1=17
GOSUB 3000
```

Il valore di un parametro può essere impostato in parecchi modi, ad esempio dai comandi READ, INPUT o FOR nonchè da una semplice assegnazione. Per visualizzare un tronco di piramide, scriveremo:

```
10 FOR X1=1 TO 18
20 GOSUB 3000
30 NEXT X1
40 STOP
```

(seguito dalla subroutine che visualizza gli ★).

Impostare questo programma (con la subroutine) e controllare che funzioni come previsto.

LE DECISIONI NEL PROGETTARE LE SUBROUTINE

Esaminiamo ora alcune delle decisioni prese mentre questo programma veniva scritto. Durante il disegno iniziale, il programmatore aveva scoperto che gli occorrevano subroutine per visualizzare un numero variabile di ★ su una riga. Egli aveva quindi deciso senza alcun motivo particolare di inserire la subroutine a 3000 e di usare X1 come parametro. In questa fase, avrebbe potuto ugualmente inserire la subroutine ovunque (ad esempio a 4500) e avrebbe potuto scegliere una diversa variabile (ad esempio N1) come parametro.

Una volta presa la decisione, comunque, le possibilità erano molto più ristrette. La subroutine ora doveva partire


```
3000REM.....
```

I richiami dovevano usare X1 come parametro ed essere scritti nella forma:

```
GOSUB 3000
```

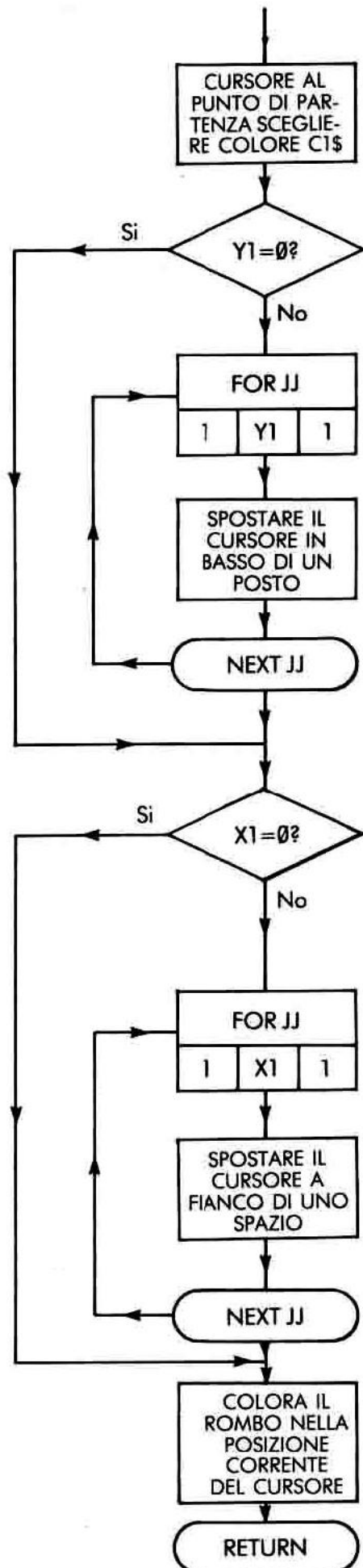
Questa è una buona illustrazione di un argomento generale: quando si inizia a disegnare un programma, c'è ampia libertà per svolgere le cose in maniere diverse; ma una volta presa una decisione, la libertà si riduce sempre più fino a che al termine risulta esserci una sola via percorribile.

USO DI PIÙ DI UN PARAMETRO

Le subroutine non sono limitate ad un parametro, ma possono usare qualsiasi numero (ragionevole) di parametri. Qui per esempio c'è una subroutine che visualizza un rombo colorato in qualsiasi posizione sullo schermo. I parametri sono:

X1: Numero degli spostamenti a destra
Y1: Numero degli spostamenti in basso
C1$: Colore del rombo

Glossario

C1$: Colore del rombo
X1, Y1: Posizione del rombo
JJ: Usato per contare i movimenti del cursore

La corrispondente codifica è:

```
2000 REM VISUALIZZA ROMBO
     COLORATO C1$ IN X1 SPAZI
     IN ORIZZONTALE SULLO SCHERMO
     E IN Y1 SPAZI IN VERTICALE
2010 PRINT " [CLR HOME] ";C1$;
2020 IF Y1=0 THEN 2060
2030 FOR JJ=1 TO Y1
2040 PRINT " [CRSR ↓] ";
2050 NEXT JJ
2060 IF X1 = 0 THEN 2100
2070 FOR JJ=1 TO X1
2080 PRINT " [CRSR ⇒] ";
2090 NEXT JJ
2100 PRINT " [CTRL] e [RVS ON]
```

```
";
2110 RETURN
```

Questa subroutine usa i comandi del cursore in stringhe per spostare il cursore sullo schermo. La 2100 colora un rombo. Sembra spaventosa quando viene scritta completamente, ma la stringa comprende soltanto nove caratteri: Reverse on (negativo attivato), reverse off (negativo disattivato) ◢ e ◣ (due volte ciascuno) e tre movimenti del cursore per arrivare dalla fine della prima riga di grafici all'inizio della seconda. I caratteri nella stringa sono esattamente quelli che si userebbero se si dovesse disegnare un rombo direttamente dalla tastiera.

Si ricorderà che nel **64** BASIC i comandi FOR seguono l'iterazione almeno una volta anche se il valore finale è meno del valore iniziale. La prova per Y1=0 è inclusa in modo che l'iterazione FOR nelle righe 2030-2050 può essere saltata del tutto se necessario. La prova per X1=0 c'è per un motivo analogo.

Per provare la subroutine ecco un programma che riempie lo schermo con rombi verdi:

```
10 PRINT" [SHIFT] e [CLR HOME] ";
20 C1$= " [CTRL] e [GRN] "
30 FOR X1=1 TO 37 STEP 3
40 FOR Y1=0 TO 21 STEP 4
50 GOSUB 2000: REM DISEGNA
     ROMBO COLORATO –C$ IN X1, Y1
60 NEXT Y1
70 NEXT X1
80 GOTO 80: REM STOP ITERAZIONE
```

ESPERIMENTO 18·2

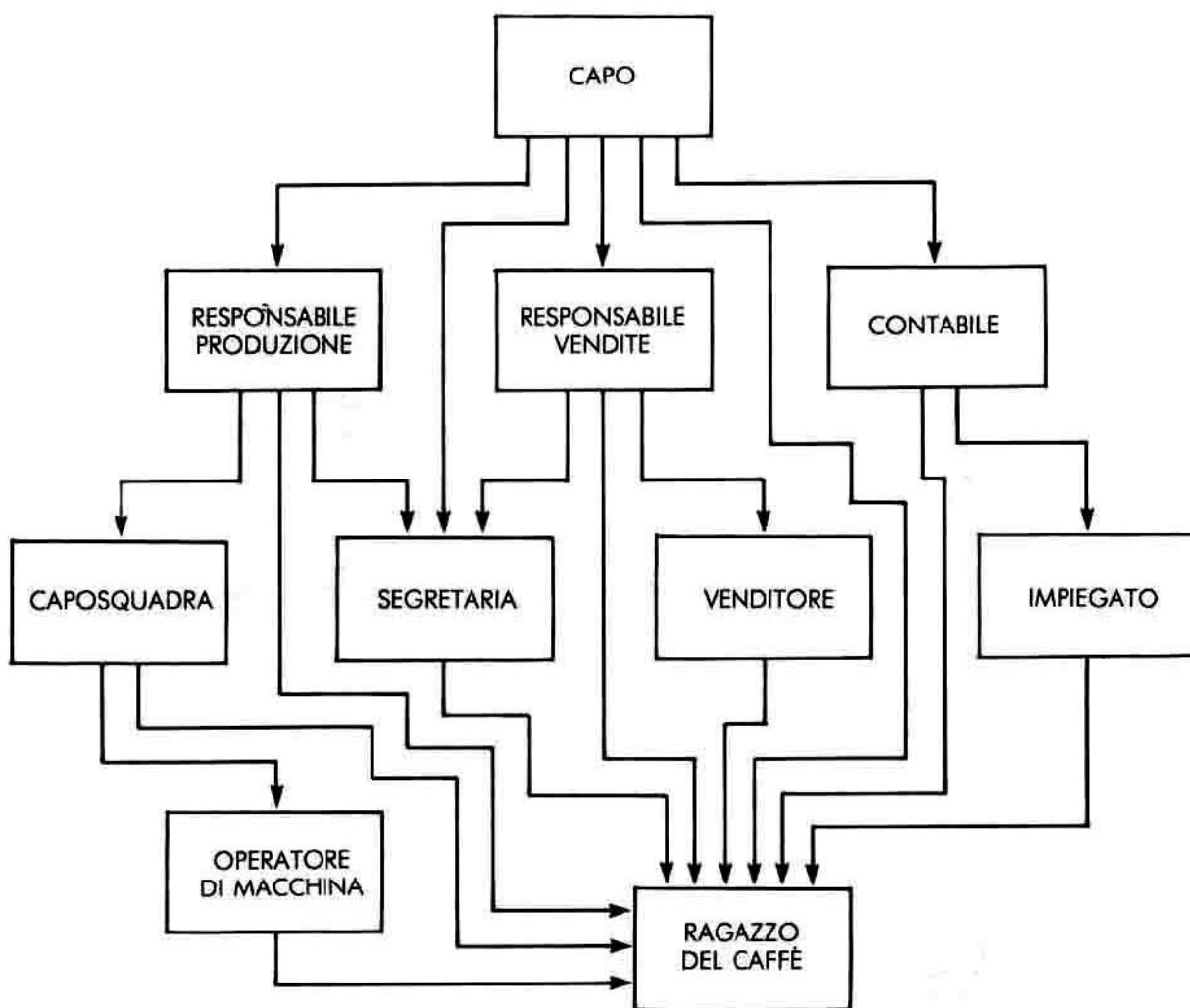
Scrivere una subroutine, iniziando alla riga 500 che disegna un "mostro" di colore C1$, 2 righe al disotto della parte superiore dello schermo e X1 spazi a partire da sinistra. Il mostro può essere semplice o complicato a piacere.

Aggiungere ora una subroutine al seguente programma che farà sì che il mostro si sposti sullo schermo da sinistra a destra:

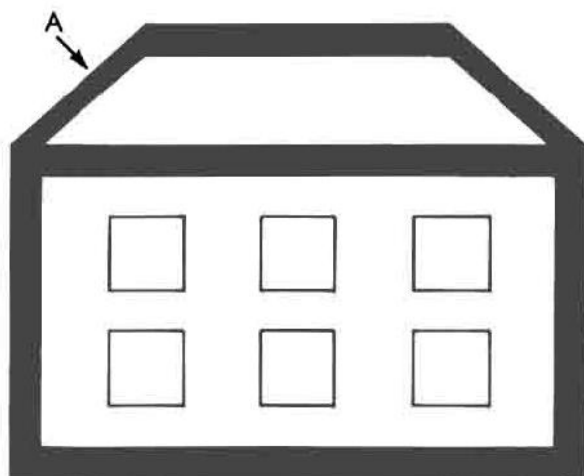
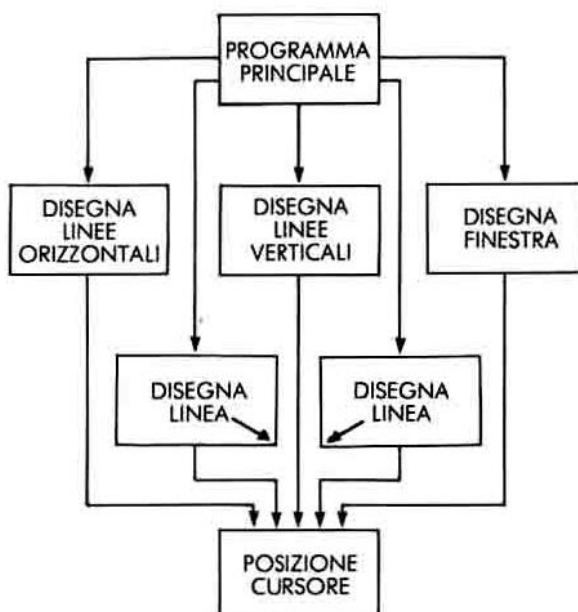
```
10 PRINT" [SHIFT] e [CLR HOME] ";
20 FOR X1 = 0 TO  35
30 C1$ =" [CTRL] e [RED] "
40 GOSUB 500: REM DISEGNA MOSTRO
     ROSSO IN X1
50 FOR T=1 TO 150: NEXT T: REM
     ATTENDI UN POCO
60 C1$ =" [CTRL] e [BLU] "
70 GOSUB 500: REM CANCELLA MOSTRO
     DISEGNANDOLO IN BLU
80 NEXT X1
90 90:REM STOP ITERAZIONE
```

Esperimento 18.2 completato	

SUBROUTINE PIÙ COMPLESSE

La maggior parte degli uffici non sono così semplici come quello che abbiamo descritto precedentemente in questa unità. Generalmente parlando, il capo è aiutato da parecchi "dirigenti", ciascuno dei quali può avere uno o più assistenti personali. Questi assistenti possono a loro volta avere propri aiutanti e così via giù giù lungo la catena di comando. Alcune persone possono eseguire il loro particolare lavoro per chiunque lo richieda; per esempio, il ragazzo del caffè deve servire chiunque. Il capo può chiedere il caffè per sè oppure può chiedere alla sua segretaria di chiederlo a suo nome. Per chiarire la struttura dei comandi, l'ufficio può avere un organigramma più o meno come questo:

Questo tipo di struttura può essere riflesso in un programma BASIC. Una subroutine può essere richiamata dal programma principale o può essere richiamata da qualsiasi altra subroutine a qualsiasi "livello". Il computer non si confonde in quanto prevede arrangiamenti speciali per memorizzare tutti i concatenamenti che gli servono per ritornare al punto giusto nel programma principale. La catasta in cui i concatenamenti sono memorizzati, non è semplicemente un richiamo di memoria ma prevede una posizione separata per ciascun "livello" di richiamo.

Caricare ed eseguire il programma denominato PICTURE dalla cassetta di nastro. Esso genera un disegno grezzo, ma riconoscibile di una casa. Il disegno è realizzato richiamando una serie di subroutine, una per ciascuna fila di finestre nell'immagine. Pertanto la subroutine alla riga 1000 disegna una riga orizzontale da sinistra a destra. La riga è lunga N1 unità e inizia alla locazione X1 spazi in orizzontale sullo schermo e Y1 righe verso il basso. Analogamente, la subroutine in 2000 disegna le righe verso il basso, la subroutine in 3000 diagonalmente verso l'alto da sinistra a destra e quella a 4000 diagonalmente verso il basso da sinistra a destra. Pertanto, per disegnare la linea contrassegnata A nell'immagine, la sequenza di richiamo è

X1=1: Y1=8: N1=5: GOSUB 3000

Listare ed esaminare il codice nelle varie subroutine. Esse iniziano tutte con un compito comune: inserire il cursore nella posizione indicata da X1. Dato che si tratta di un lavoro ben definito, è comodo trasformarla in una subroutine a sè. La "struttura di potere" del programma è pertanto:

Ora cancellare i comandi da 10 a 140 e usare le subroutine per disegnare un'immagine a scelta. Potrebbe trattarsi di un castello o di una fabbrica con una ciminiera. È inoltre possibile definire una nuova subroutine per disegnare finestre con archi e disegnare una chiesa.

Esperimento 18.3 completato

Le subroutine sono un argomento importante e se ne continuerà la discussione nella succesiva unità.

UNITA':19

Altro sulle subroutine	PAGINA 185
Specifica delle subroutine	185
Subroutine per semplificare le frazioni	185
Programma di gestione	187
ESPERIMENTO 19.1	188
Robustezza delle subroutine	188
Limiti del campo di un parametro	188
ESPERIMENTO 19.2	190
Convenzioni di denominazione delle subroutine	191
ESPERIMENTO 19.3	193
ESPERIMENTO 19.4	193

ALTRO SULLE SUBROUTINE

In questa unità si continuerà lo studio delle subroutine. Il segreto per scrivere robusti e utili programmi e per farli lavorare rapidamente, è una raccolta di tecniche dette "software engineering". Queste tecniche non sono solitamente citate nei manuali introduttivi in quanto sono generalmente considerate strumenti per professionisti. Non vale quindi la pena di imparare a programmare scorrettamente quando è altrettanto facile farlo nella maniera corretta.

SPECIFICHE DELLE SUBROUTINE

Un'idea vitale nel software engineering è la *specifica delle subroutine*. Si tratta di una descrizione esatta di cioè che una subroutine fa e di come (e cioè attraverso quali parametri) essa comunica con il programma principale. La specifica della subroutine non dice nulla sul meccanismo interno della subroutine stessa o sul modo in cui essa svolge il compito che è predisposta per fare.

Le specifiche delle subroutine servono a due scopi diversi. Innanzitutto possono essere stampate in un catalogo di subroutine in modo che altri programmatori possano scegliere utili subroutine per i rispettivi programmi ed effettuare tutti gli arrangiamenti pratici per richiamarle. Secondo, una specifica di subroutine dà un saldo punto di partenza al programmatore che scrive la subroutine stessa. Egli può scriverla in qualsiasi modo a suo piacere purchè esegua esattamente ciò che la specifica dice. Notare l'ordine delle cose: la specifica prima del programma. La sola eccezione è che talune voci possono dover essere aggiunte alla specifica dopo che la subroutine è stata scritta.

Vediamo un esempio. Si supponga di scrivere un programma per aiutare i bambini ad eseguire le somme con le frazioni tipo "1/6 + 1/32" o "5/8 × 2/3". Se le somme sono generate casualmente, in qualche punto il programma dovrà elaborare le proprie risposte alle domande che esso pone. Si ricorderà che lavorando con le frazioni era possibile ottenere risposte nei termini inferiori. Ad esempio si poteva ottenere:

$$\text{"}1/6 + 1/3 = \frac{1+2}{6} = 3/6\text{" o "}5/8 \times 2/3 = 10/24\text{"}$$

Le frazioni "3/6" e "10/24" non sono sbagliate ma devono essere semplificate prima di poter essere accettate come completamente corrette.

Il lavoro per semplificare le frazioni è un compito autonomo, chiaramente adatto per farne una subroutine. Questa subroutine sarà diversa da quelle nell'unità 18 per una cosa molto importante: essa prenderà i parametri dal programma principale, eseguirà un calcolo e darà i risultati al *programma principale*, non visualizzerà cioè nulla sullo schermo (salvo eventualmente in caso di emergenza), nè richiederà alcun input dall'utente. Per quanto riguarda la subroutine, il programma principale è il mondo esterno.

Iniziamo identificando e denominando i parametri. Per fare questo lavoro la subroutine ha bisogno di una *coppia* di numeri (ad esempio 3 e 6 o 10 e 24) che sono rispettivamente il "numeratore" e il "denominatore" della frazione che si sta cercando di semplificare. Sceglieremo A1 e B1 per rappresentare i valori originali. A1 e B1 sono detti *parametri di input*, quantunque l'input avvenga nel programma principale e non (almeno direttamente), dall'utente.

Il risultato della subroutine è un'altra coppia di numeri tipo 1 e 2 o 5 e 12. Faremo in maniera che la subroutine ritorni a questi valori in C1 e D1. Come ci si potrebbe aspettare, C1 e D1 sono detti parametri di *output*, quantunque non venga eseguita alcuna stampa (PRINT) ad opera della subroutine.

Infine, decideremo a caso di inserire il primo comando della subroutine in 5500. Questo numero non entra in conflitto con qualsiasi altra subroutine nella nostra raccolta.

Possiamo ora scrivere una specifica formale:

Specifica di subroutine provvisoria

Scopo: Semplificare le frazioni ai loro minimi termini

Numeri di riga: da 5500 a [ ]

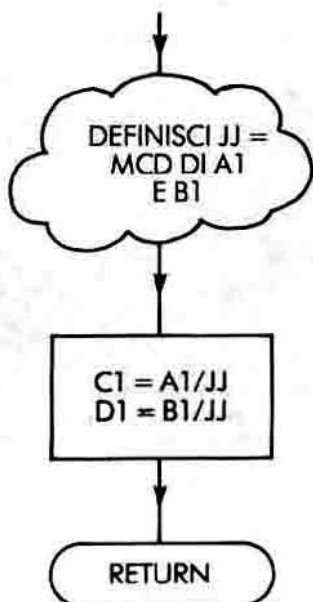
Parametri:
Input A1 (numeratore frazione)
B1 (denominatore frazione)
Output C1 (numerat. fraz. semplificata)
D1 (denomin. fraz. semplificata)

Variabili locali: [ ]

Notare che la specifica è provvisoria e contiene due caselle vuote. Una è prevista per l'ultima riga della subroutine e l'altra è intesa per una lista delle variabili usate dalla subroutine stessa. Queste caselle non possono essere riempite fino a che la subroutine non è stata scritta.

SUBROUTINE PER SEMPLIFICARE LE FRAZIONI

Ora torniamo alla subroutine. Per semplificare una frazione occorre per prima cosa trovare il cosiddetto "massimo comun divisore" e quindi dividere per esso numeratore e denominatore. Per esempio, il MCD di 10, 24 è 2 e 10/24 in termini ridotti è pertanto 5/12.

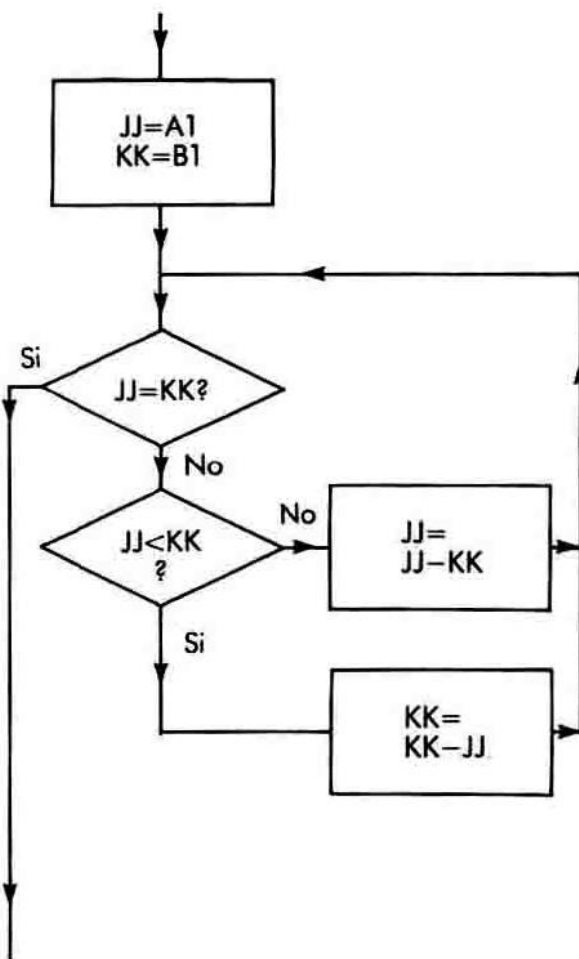
Questo processo viene facilmente riportato su uno schema di flusso. Non conosciamo ancora come funziona l'MCD dei due numeri cosicchè useremo una nuvoletta:

Glossario

A1/B1: Frazione da ridurre ai minimi termini
C1/D1: Risultato

Un modo semplice per trovare l'MCD di due numeri è detto algoritmo di Euclide. Iniziando con i due numeri sottrarre il più piccolo dal più grande fino che i due sono identici: questo valore è il cosiddetto MCD. Iniziando con 10 e 24 si ottiene:

24	10	
		Togliere 10 da 24
14	10	
		Togliere 10 da 14
4	10	
		Togliere 4 da 10
4	6	
		Togliere 4 da 6
4	2	
		Togliere 2 da 4
2	2	
		2=2, cosicchè l'MCD di 10 e 24 è 2

Questo processo può essere riportato su uno schema di flusso come segue:

Nota: Usiamo variabili locali JJ e KK in modo da non danneggiare i valori di A1 e di B1.

Glossario

A1, B1: numeri di cui si vuole il massimo comun divisore
JJ,KK: variabili locali (risultato in JJ)

La subroutine potrebbe apparire come segue:

```
5500 REM RIDURRE FRAZIONE A1/B1 AI
     SUOI MINIMI TERMINI
5510 REM RISULTATO IN C1/D1 LOCALI
     =JJ,KK
5520 JJ=A1 : KK=B1
5530 IF JJ = KK THEN 5560
5540 IF JJ < KK THEN KK=KK-JJ: GOTO
     5530
5550 JJ=JJ-KK : GOTO 5530
5560C1=A1/JJ : D1=B1/JJ
5570 RETURN
```

Possono ora essere inseriti gli ultimi dettagli rimanenti della specifica.

Specifica della subroutine

Scopo: Semplificare le frazioni ai loro minimi termini

Numeri di riga: da 5500 a 5570

Parametri:
Input A1 (numeratore frazione)
B1 (denominatore frazione)
Output C1 (numerat. fraz. semplificata)
D1 (denomin. fraz. semplificata)

Variabili locali: JJ, KK

PROGRAMMA DI GESTIONE

Per controllare la subroutine abbiamo bisogno di un programma "di gestione". Questo è il "programma principale" più semplice che possiamo costruire che prova la subroutine in ogni modo ragionevole.
La specifica della subroutine risulta estremamente utile per scrivere il programma di gestione. Essa ci dice:
(a) di inserire i parametri di input in A1 e B1
(b) di chiamare la subroutine in 5500
(c) di osservare i risultati in C1 e D1
(d) di evitare di usare le righe da 5500 a 5570 per qualsiasi altro scopo
(f) di non usare JJ e KK nel programma principale.

In generale, se una specifica non comprende le informazioni che occorrono per scrivere un programma di gestione, la specifica è incompleta.
Un adatto programma di gestione è il seguente:

```
10 INPUT "FRAZIONE"; A1,B1
20 GOSUB 5500
30 PRINT "RISULTATO="; C1;"/"; D1
40 GOTO 10
```

Alcune prove producono quanto segue

```
RUN
FRAZIONE? 33,67
RISULTATO = 33/67
FRAZIONE? 33, 69
RISULTATO = 11/23
FRAZIONE? 12345, 23456
RISULTATO = 12345/23456
FRAZIONE? 10, 24
RISULTATO = 5/12
FRAZIONE? 3, 6
RISULTATO = 1/2
. . . .
```

Questi risultati sembrerebbero esatti e per il momento accetteremo la subroutine come corretta.

ESPERIMENTO 19.1

Scrivere un programma che consenta all'utente di battere due frazioni (ad esempio p/q e s/t) e di visualizzare il risultato semplificato della loro somma. Per esempio:

```
PRIMA FRAZIONE? 3, 8          (significa 3/8)
SECONDA FRAZIONE? 5,12        (significa 5/12)
SOMMA = 19/24
```

SUGGERIMENTO: $p/q + s/t = \frac{p\star t + q\star s}{q\star t}$
esempio:

$$3/8+5/12=\frac{3\star 12+5\star 8}{8\star 12}$$

Porre A1= P★T+Q★S,B1=Q★T e quindi richiamare la subroutine di semplificazione.

Esperimento 19.1 completato	

Robustezza delle subroutine
È interessante confrontare i programmi scritti da professionisti con quelli prodotti da amatori inesperti. Un programma di un professionista ha due caratteristiche essenziali:
(a) È *robusto*. Esso non dà mai risposte sbagliate e non "si rompe" mai visualizzando errori non previsti o bloccandosi in un'iterazione se l'utente fornisce l'informazione scorretta.
(b) E *adattabile*. Il programma è costruito e documentato con schemi di flusso, glossari, specifiche di subroutine e commenti (REM) in modo che qualsiasi programmatore competente possa facilmente modificarlo per adattarlo ad una nuova esigenza.

Per contro, il programma di un dilettante è *fragile*. Esso solitamente funziona finchè viene usato da colui che lo scrive, pronto ad intervenire se viene immesso qualsiasi input scorretto. Esso non è per nulla documentato e probabilmente non ha alcuna struttura discernibile. Pochi mesi dopo averlo scritto il programmatore dimentica come funziona e nessuno riesce a capirci più nulla. In queste condizioni, la maggior parte dei tentativi volti a correggere eventuali errori o migliorare il programma non fa che peggiorare le cose. La robustezza di una catena si misura dal suo anello più debole.
Allo stesso modo un programma è affidabile nella misura in cui lo è la subroutine meno affidabile. Uno dei concetti base del software engineering è che ogni subroutine dovrebbe essere *perfetta*, o almeno tanto perfetta quanto è possibile farla. Dovrebbe essere *impossibile* per una subroutine fare qualcosa di sbagliato senza almeno darne una segnalazione.

LIMITI DEL CAMPO DI UN PARAMETRO

A pagina 178 c'era una semplice subroutine con un unico parametro X1, che visualizzava un numero di asterischi su una riga. In quell'unità si assumeva con ottimismo che tutto fosse corretto; ma esaminiamolo ora più da vicino. Eccolo qui di nuovo unitamente ad un programma di gestione:

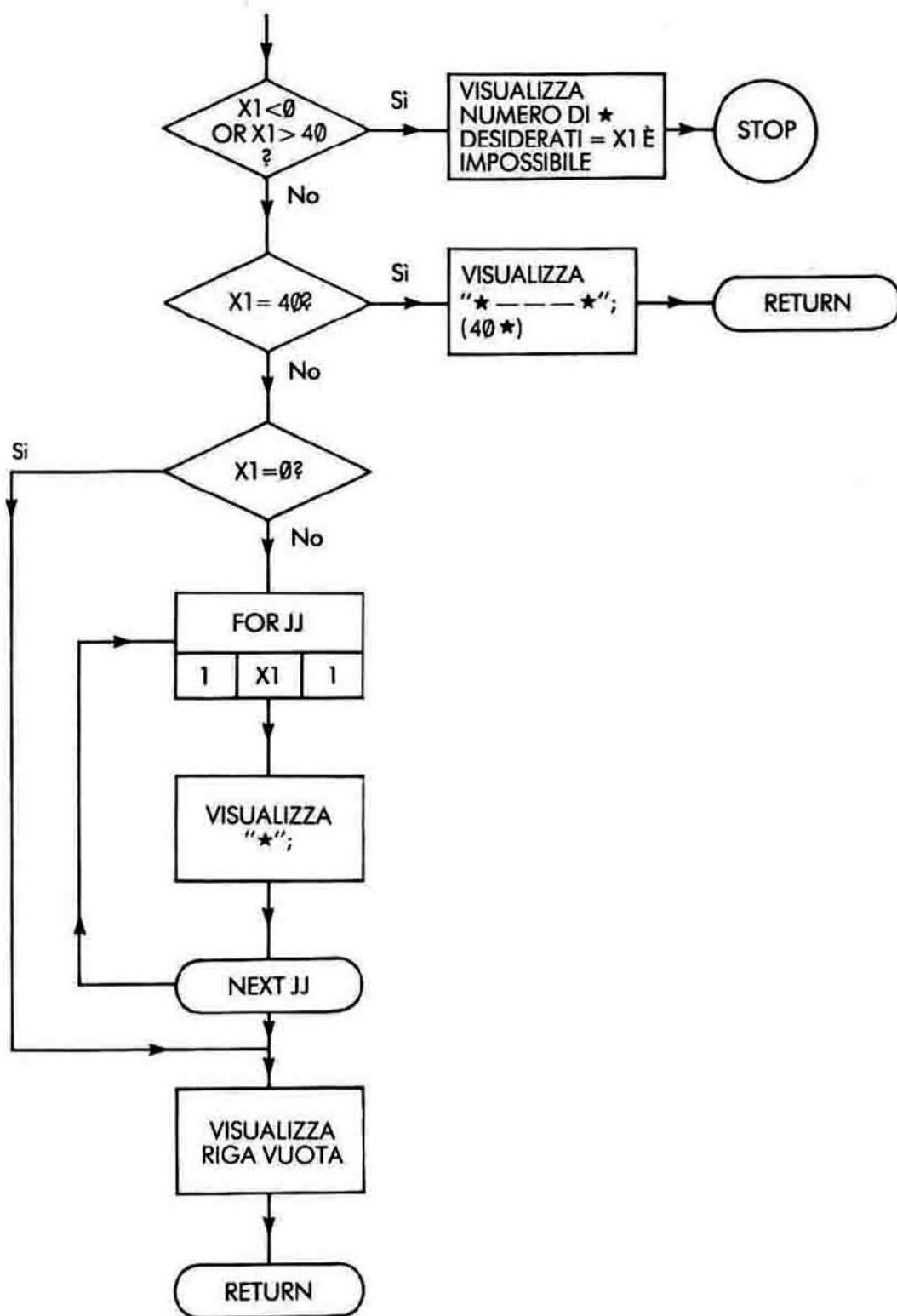
```
  10 UNPUT"NUMERO ASTERISCHI"; X1
  20 GOSUB 3000
  30 GOTO 10
3000 REM VISUALIZZA NUMERO ★
     DATI IN X1 SU UNA RIGA
3010 FOR JJ = 1 TO X1
3020 PRINT "★";
3030 NEXT JJ
3040 PRINT
3050 RETURN
```

Innanzitutto notiamo che la variabile locale JJ non è citata nel commento (REM) alla riga 3000 ed accettiamo ciò come un errore piccolo ma genuino. Ora iniziamo la prova. La subroutine sembra funzionare bene per X1=1,3,6 e così via. Con fiducia proviamo altri numeri:

39: Funziona correttamente.
40: Dà 40 asterischi e una riga vuota in più! Se si usasse la subroutine per disegnare un grafico, ciò distruggerebbe il suo aspetto.
41: Ciò non *dà* una riga con 40 asterischi; visualizza una riga di 40 e una seconda riga con un asterisco.
0: Ci si aspetta una riga vuota; per contro si ottiene una riga con un asterisco.
−3: Questo è un valore privo di significato cosicchè ci si dovrebbe aspettare che la subroutine dia un avvertimento del tipo "cosa significa?". Per contro dà una riga con un asterisco, esattamente come se si fosse impostato X1=1.

Sta diventando penosamente chiaro che il programma non è conforme alla sua specifica. Abbiamo bisogno di qualche modifica.
Innanzitutto una riga può contenere soltanto tra 0 e 40 asterischi cosicchè dobbiamo fare in modo che la subroutine *respinga* qualsiasi valore aldifuori di questo campo. Un programma richiamante che fornisse un valore fuori campo per il parametro X1, sarebbe presumibilmente in errore cosicchè è appropriato fare in modo che la subroutine visualizzi un messaggio di avvertimento e si fermi.
Secondo, la subroutine ha bisogno di predisposizioni speciali per i valori estremi 0 e 40. Nella versione originale 0 era stato gestito scorrettamente in quanto il comando FOR in BASIC è sempre eseguito almeno una volta, *anche* per i comandi tipo

```
FOR JJ = 1 TO 0
```

All'altro estremo, 40 portava ad una riga extra indesiderata, in quanto viene *automaticamente* inserita una nuova riga dopo il 40esimo carattere di qualsiasi riga.
Tenendo nota di questi punti, ecco uno schema di flusso e un programma revisionati:

Glossario

X1: Il numero di asterischi da visualizzare
JJ: Contatore asterischi

```
3000 REM VISUALIZZA ASTERISCHI DATI
     IN X1 SU UNA RIGA.
3005 REM DEVE ESSERE FRA 0-40.
     VARIABILE=JJ
3010 IF X1<0 OR X1>40 THEN PRINT
     "NUMERO★DESIDERATI=";X1:
     GOTO 3090
3020 IF X1=40 THEN 3100
3030 IF X1=0 THEN 3090
3040 FOR JJ=1 TO X1
3050 PRINT"★";
3060 NEXT JJ:PRINT:RETURN
3090 PRINT"IMPOSSIBILE":STOP
3100 PRINT"★★★★★★★★★★★★★★★★
★★★★★★★★★★★★★★★★★★★★
★★★★;:PRINT:RETURN
```

Questo illustra tre fatti importanti:

(a) Dove i parametri di una subroutine possono assumere soltanto un range limitato di valori, il buon software engineering richiede che la subroutine debba controllare che ogni valore sia entro il range e segnalare qualsiasi discordanza.

(b) Quando una subroutine viene testata, è particolarmente importante provare i valori estremi ammessi (ad esempio 0 e 40) dato che è qui dove spesso si annidano gli errori.

(c) Le subroutine che sono correttamente strutturate e funzionano sicuramente in tutti i casi, sono solitamente più lunghe delle loro controparti più semplici.

ESPERIMENTO 19·2

(a) Il programma che segue si propone di visualizzare le registrazioni di 11 giocatori di cricket nella forma di un "istogramma" o grafico a barre, dove ciascuna fila corrisponde ad un giocatore e ciascuna stella ad una porta. Eseguire il programma con la vecchia e la nuova versione della subroutine iniziando alla riga 3000 e osservare la differenza:

```
10 REM ISTOGRAMMA BATTUTE
20 PRINT" SHIFT e CLR HOME"
30 FOR J = 1 TO 11
40 READ X1: GOSUB 3000
50 NEXT J
60 STOP
100 DATA 3,7,25,40,5,0,4,1,0,40,15
```

(b) Tornare alla subroutine indicata a pagina 186 e provarla di nuovo più a fondo. Cosa succede se A1 o B1 sono 0 o negativi (ad esempio −5)?
A1 o B1 sono decimali (ad esempio 3.143)?

Queste prove convinceranno (se non lo si sapeva già) che l'algoritmo di Euclide funziona soltanto per numeri interi positivi.

Disegnare una versione correttamente strutturata della subroutine ricordando che:

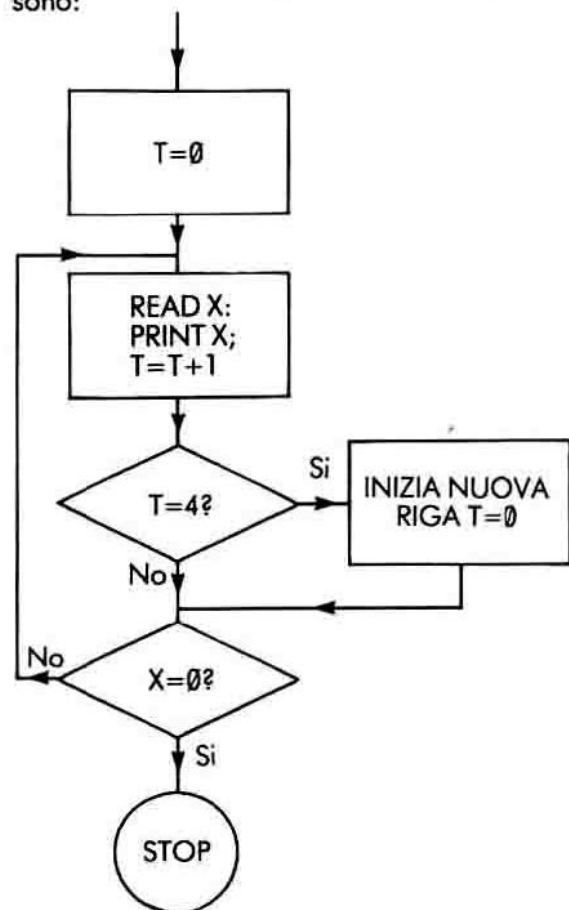
(a) Non è sensato per A1 e B1 avere valori frazionari (quantunque ciò potrà accadere). Se un numero X è un *intero*, l'espressione X=INT(X) è vera.

(b) Non è sensato per B1 avere qualsiasi valore minore di 1.

(c) È sensato che A1 sia 0 o un numero negativo; ciò potrebbe avvenire durante la sottrazione di due frazioni. Se A1=0, il valore del risultato dovrebbe essere 0/1, indipendentemente da B1. Se A1 è negativo, la subroutine dovrebbe ricordare il fatto, usare un numero positivo nell'algoritmo di Euclide e cambiare il segno di C1 immediatamente prima che il risultato venga fornito.

Esperimento 19.2 completato	

CONVENZIONI DI DENOMINAZIONI NELLE SUBROUTINE

Nelle discussioni sulle subroutine abbiamo introdotto una convenzione di denominazione: A1, B1, ...Z1 per i parametri e AA...ZZ per le variabili locali. Non ci sono regole fisse su questi nomi in BASIC e si troveranno numerosi programmi che non osservano la convenzione, ma vale la pena di rispettarla in quanto protegge contro alcuni degli errori più sottili che possono verificarsi nei grandi programmi.
In pratica è abbastanza comune che i programmi non riescano a funzionare in quanto il valore di qualche variabile importante è stato danneggiato da una subroutine. Ecco un semplicissimo esempio.
Si consideri un programma che abbia una lista di numeri nelle sue istruzioni DATA, supponendo che ne legga e ne visualizzi quattro per riga. L'ultimo numero, che agisce da terminatore, è Ø.
Per organizzare la struttura si usa una variabile T per contare la quantità di numeri già su una riga. Ogni volta che viene visualizzato un numero, aumentiamo T di 1. Quando esso raggiunge 4, iniziamo una nuova riga e riportiamo T a zero.
Lo schema di flusso generale e il programma sono:

Glossario

X: Numero corrente
T: Numeri di elementi sulla riga corrente

```
10 T=0
20 READ X
30 PRINT X;
40 T=T+1
50 IF T=4 THEN PRINT:T=0
60 IF X <> 0 THEN 20
70 STOP
80 DATA 15,23,40,11,37,51,99
90 DATA 33,12,89,53,17,20,0
```

Se si imposta questo programma, si troverà che funziona perfettamente.
Si supponga ora che un anno dopo, quando non si ricordano più i dettagli del programma, si decida di fare una modifica: si vuole che il computer emetta un segnale acustico ogniqualvolta visualizza un nuovo numero. Nel catalogo della subroutine si trova:

Specifica della subroutine

Scopo: Creare un segnale acustico seguito da mezzo secondo di silenzio

Righe: 2000-2050

Parametri: Nessuno

Questa subroutine sembra interamente adatta. Si aggiunga il testo al programma:

```
2000 REM SUBROUTINE PER CREARE
     SUONO PIP
2010 VV=212*256
2020 POKE VV+4,0
2030 POKE VV+24,15:POKE VV,0
2040 POKE VV+1,80:POKE VV+5,9
2050 POKE VV+4,33
2060 FOR T=1 TO 500:NEXT T
2070 POKE VV+4,0
2080 RETURN
```

e si inserisca un nuovo comando:

```
25   GOSUB 2000
```

Sfortunatamente il programma non funziona più; i segnali acustici si verificano come si prevedeva ma lo schema è andato tutto all'aria. Il motivo è che la variabile di controllo dello schema T è stata modificata dalla subroutine. Ciò non sarebbe accaduto se il creatore della subroutine avesse seguito almeno in parte le convenzioni. Se avesse chiamato la sua variabile locale TT (invece di T), non la si sarebbe usata nel programma principale; oppure se avesse citato 'T' come variabile locale nella specifica di subroutine, ci sarebbe stato l'avvertimento.
In questo esempio, il fatto che il programma modificato fosse difettoso, era immediatamente ovvio. Talvolta l'errore è così facile da scoprire; esso semplicemente porta a delle risposte sbagliate. Si consideri questo programma che immette una serie di numeri e ne visualizza il totale.

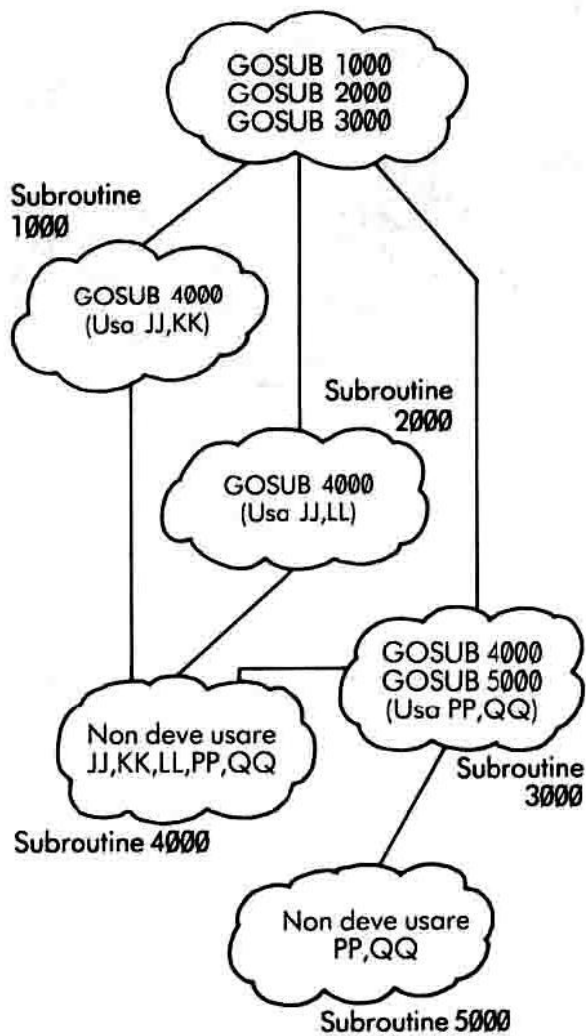
```
10 INPUT "QUANTI NUMERI"; N
20 T=0
30 FOR J=1 TO N
40 INPUT X
50 T=T+X
60 NEXT J
70 PRINT "TOTALI"; T
80 STOP
```

Glossario

N: Numero dei numeri
T: Totale mobile
X: Numero successivo da leggere
J: Conteggio dell'immissione di numeri

Questo funziona bene. Si supponga ora che il programmatore decida di migliorare il tutto facendo in modo che il programma emetta un segnale acustico ogni volta che accetta un numero. Egli aggiunge la subroutine del catalogo e la suddivide in due comandi extra:

```
  15 GOSUB 2000
e 45 GOSUB 2000
```

Il programma è ora molto più soddisfacente da usare: esso suona come un moderno registratore di cassa. Sfortunatamente ora asserisce che sommando 247,37,12,93,52 e 39 si ottiene 540. Questa risposta sembra ragionevole ma effettivamente è sbagliata! Se non si notasse l'errore e si usasse il programma nella propria azienda, si potrebbe terminare con un "fallimento assistito dal computer". L'errore è abbastanza facile da notare una volta che si sa dov'è, ma il fatto triste è che ci sono numerosi errori analoghi nascosti ovunque nei programmi, completamente insospettati fino a che non fanno crollare ponti, non fanno morire i pazienti e non fanno precipitare i razzi in mare.
Se si scelgono le convenzioni di denominazione, è possibile solitamente evitare questo tipo di errore. Il programma principale non deve *mai* usare variabili "con la doppia lettera" che sono riservate per le variabili locali nelle subroutine e si deve usare soltanto la forma "lettera −1" tipo A1 o B1 per i parametri.
Se il programma usa più di una subroutine, occorre fare in modo che esse si adattino l'una all'altra. Chiaramente, tutte le subroutine devono usare diversi numeri di riga e se necessario occorrerà variarne di conseguenza uno o più.
Quando due o più subroutine vengono richiamate "allo stesso livello" (ad esempio potrebbero essere entrambe richiamate nel programma principale), esse possono usare sicuramente gli stessi parametri e le variabili locali. Se le subroutine sono a livelli diversi e una di esse "richiama" l'altra, devono usare variabili locali e parametri diversi. Tutto ciò è reso chiaro dal seguente diagramma:

Le unità che seguono faranno ampio uso delle subroutine di tutti i tipi. Munirsi quindi di un quaderno a fogli mobili e iniziare la propria libreria di subroutine. Ciascuna voce dovrebbe avere quattro elementi di documentazione:

Specifica
Schema di flusso
Glossario
Testo origine (e cioè il programma stesso)

ESPERIMENTO 19·3

Progettare, scrivere e documentare una subroutine che prende tre numeri come parametri e fornisce il valore del maggiore come risultato. Scrivere un adatto programma di gestione e provare la subroutine a fondo il più possibile.

Esperimento 19.3 completato	

ESPERIMENTO 19·4

Il file denominato "BIGLETTERS64" sulla cassetta di nastro è una subroutine che consente all'utente di battere una lettera o un carattere e di visualizzarlo quattro volte più grande.
Qui di seguito sono indicate le specifiche dell'intera subroutine. Studiarla e scrivere un programma che crei una riga di testo a caratteri cubitali.

Specifiche della subroutine

Scopo: Visualizzare i caratteri del **64** quattro volte più grandi

Numeri di riga: da 8000 a 8200

Parametri: Input: La subroutine deve essere richiamata una volta per ciascun carattere. Il carattere da visualizzare deve essere fornito sotto forma di stringa di 1 carattere di nome A1$.
Output: A1$ (convertito in quattro volte il suo corpo normale).

Variabili locali: AA,BB,JJ,KK,LL,MM,NN,QQ

Note: (i) QQ non deve essere usato aldifuori della subroutine per qualsiasi motivo.
Note: (ii) La subroutine gestisce tutti i caratteri stampabili nelle serie "non shiftata" "shiftata" e "Commodore". Essa accetta anche e interpreta BLK, WHT, READ, CYN, PUR, GRN, BLU, YEL, RVS ON, RVS OFF, CLR HOME e RETURN. Altri tasti tipo DEL e i tasti di controllo del cursore sono ignorati.

Esperimento 19.4 completato	

MATRICI

Una tipica classe scolastica potrebbe essere composta dai seguenti nomi:

ADAMI
BONINI
CARLETTI
FINETTI
MAGNANI
MONETTI
SILVESTRI
TOMMASI
VALENTI
ZILIOLI

I fortunati sono quelli che hanno il cognome che inizia con una delle prime lettere dell'alfabeto. Ciò significa che sono i primi ai quali viene offerta la scelta di un tavolo, ecc. e non devono aspettare a lungo ad essere interrogati. Al povero Zilioli non viene data alcuna scelta ed è sempre l'ultimo in ogni coda. Talvolta, l'insegnante inizia la lettura dei nomi alla rovescia e Zilioli sarà il primo, ma ciò non risolve il problema di Zilioli.
Pensiamo di scrivere un programma per capovolgere una lista di nomi. Si vuole che l'utente batta l'elenco dei nomi di una classe, un nome per ogni riga e quindi lo legga alla rovescia dallo schermo in ordine invertito. Così a prima vista non sembrerebbe un compito molto difficile rispetto a quelli che sono stati già programmati e tuttavia la soluzione è elusiva. Il meglio che si può fare è di trovare in anticipo quanti nomi ci saranno e quindi scrivere un lungo e complicato programma usando una diversa variabile per ciascun nome. Per esempio, se ci sono quattro nomi nella lista, scegliamo le variabili A$, B$, C$ e D$ e scriviamo quanto segue:

```
 10 PRINT "IMMETTI NOMI ALLIEVI"
 20 INPUT A$
 30 INPUT B$
 40 INPUT C$
 50 INPUT D$
 60 PRINT "ORDINE INVERTITO="
 70 PRINT D$
 80 PRINT C$
 90 PRINT B$
100 PRINT A$
110 STOP
```

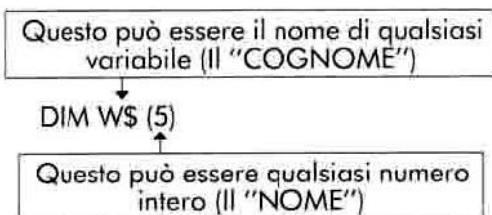
Questo programma darà una lista esattamente di quattro nomi, ma non può essere adattato per funzionare con qualsiasi altro numero di nomi, senza aggiungere (o cancellare) alcun comando. Se la classe avesse — ad esempio — 30 allievi, occorrerebbe scrivere un programma speciale con 30 variabili e 63 comandi. La scrittura del programma sarebbe come un castigo scolastico e converrebbe scrivere l'elenco manualmente.
Fortunatamente il BASIC ha un importante meccanismo che aiuta a superare questa difficoltà: si tratta della funzione *matrice*.

LE ISTRUZIONI DI DIMENSIONE

Una matrice è una *famiglia* di variabili che usa in comune il "cognome" ma che ha singoli "nomi" detti indici. Se si vuole usare tale famiglia, lo si dice normalmente al computer in uno speciale comando detto istruzione "DIM" (dimensione) perciò:

Questo può essere il nome di qualsiasi variabile (Il "COGNOME")

```
DIM W$ (5)
```

Questo può essere qualsiasi numero intero (Il "NOME")

Ciò dice al **64** di accantonare spazio per una famiglia di variabili stringa denominata W$. Ci sono sei di queste variabili e i loro nomi completi sono:

W$(0)
W$(1)
W$(2)
W$(3)
W$(4)
W$(5)

L'indice è il numero tra parentesi che segue il nome della famiglia della matrice. La prima variabile ha un indice di 0, cosicchè il numero di variabile nella famiglia è sempre *uno in più* del numero nell'istruzione DIM. È talvolta comodo dimenticarsi della presenza della variabile con indice 0 e usare soltanto le variabili che hanno indici che iniziano con 1.

USO DELLE VARIABILI MATRICE

Nella maggior parte dei casi i membri di una famiglia di variabili sono come le normali variabili. È possibile includerli nelle espressioni, stamparli, leggerli e assegnare loro dei valori. Se il nome della famiglia termina con $, ciascun membro può contenere una stringa; altrimenti ciascun membro contiene un numero.
Per illustrare questi punti ecco alcuni comandi BASIC leciti. Sia chiaro che essi non intendono formare un programma sensato!

```
DIM N(20):REM DICHIARA UNA MATRICE
      DI 21 ELEMENTI DA N(0) a N(20)
N(5) = N(3) + 5
PRINT (N1); N(2); N(3)
INPUT N(2)
IF N(12) = N(17) THEN 150
```

Notare che una cosa che non è possibile fare è di usare un membro di una famiglia, un "elemento di matrice" come è spesso chiamato, come variabile controllata in un comando FOR.

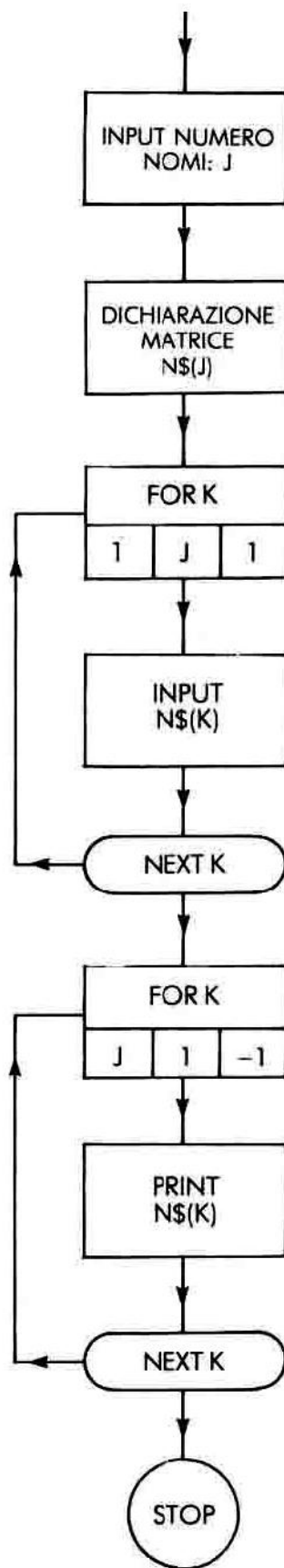
```
FOR N(5) = 1 TO 17  )
. . .               }  non è ammesso
NEXT N(5)           )
```

Finora non abbiamo detto nulla che sembra aiutarci con il problema di invertire l'elenco dei nomi. Ecco il punto chiave:

> L'indice di un elemento di matrice può essere un'espressione che viene elaborata quando il programma viene eseguito.

Si pensi a qust'idea per un momento e si veda se è possibile trarre alcune implicazioni prima di leggere. Si consideri un comando che fa parte di un'iterazione per cui viene eseguito parecchie volte ripetutamente. Se il comando include un riferimento ad un elemento di matrice, è possibile scegliere un'espressione indice in modo che venga usato un elemento diverso ogni volta che viene eseguita l'iterazione!
È possibile ora scrivere una soluzione molto più soddisfacente al problema originale. Il solo requisito extra è di chiedere all'utente di iniziare dando il numero dei nomi della lista.

Glossario

J: Numero dei nomi
Da N$(1) a N$(N): Lista dei nomi
K: Nome del contatore e indice a N$

Le corrispondenti istruzioni possono essere scritte nel modo seguente:

```
10 INPUT "QUANTI NOMI"; J
20 DIM N$(J)
30 FOR K = 1 TO J
40 INPUT N$(K)
50 NEXT K
60 FOR K = J TO 1 STEP -1
70 PRINT N$(K)
80 NEXT K
90 STOP
```

Questo semplice programma ha ottenuto la generalizzazione che era assente dai precedenti tentativi. Esso funzionerà per qualsiasi numero di nomi da 1 in su. Provare a impostarlo e a eseguirlo. Notare che la matrice N$ non è dichiarata fino a che il programma non "sa" quanti elementi deve avere. Quindi (dimenticandosi di N$(0)), gli viene attribuito esattamente il numero corretto di elementi.

Una cosa che non si deve mai fare è dichiarare la stessa matrice più di una volta. Occorre evitare cioè sequenze del tipo:

```
30 DIM A(50)
...
70 DIM A(50)
```

ma ci si deve inoltre assicurare di non inserire la dichiarazione di matrice all'interno di un'iterazione. Per esempio, se si tentasse di trasformare il semplice programma per invertire la lista in un'iterazione, inserendo

```
90 GOTO 10
```

si otterrebbe un errore la seconda volta che esso tenta di eseguire il comando DIM alla riga 20.

ESPERIMENTO 20·1

Si immagini che la dimensione della classe sia troppo grande e che di conseguenza l'insegnante non possa contare il numero correttamente. Scrivere una versione del programma di inversione della lista, che cerchi lo speciale terminatore "ZZZZ" all'estremità, invece di chiedere il numero all'inizio. Per esempio, se l'input fosse

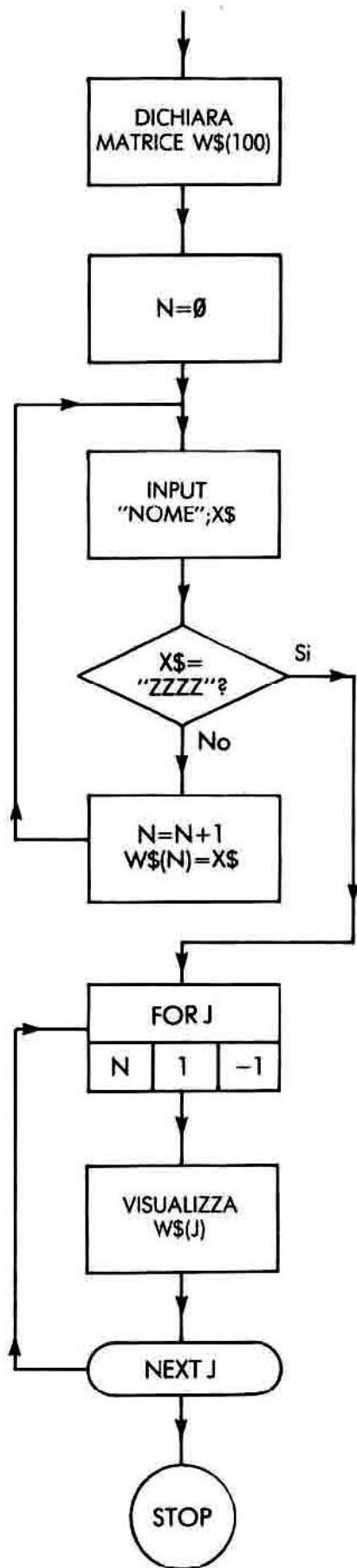
```
CRANACH
DURER
MICHELANGELO
TURNER
ZZZZ
```

l'output sarebbe

```
TURNER
MICHELANGELO
DURER
CRANACH
```

Suggerimenti: usare il seguente schema di flusso:

Glossario

W$(100): Matrice per i nomi (massimo 100)
N: Contatore dei nomi
X$: Nome corrente
J: Indice di W$

Esperimento 20.1 completato

ULTERIORI USI DELLE MATRICI

Come si è visto da questo esempio, uno dei principali vantaggi della matrice è la possibilità di avere una tabella di stringhe (o di numeri) e di fare riferimento ai suoi elementi in qualsiasi momento e in qualsiasi ordine. Ciò è spesso utile. S'immagini di scrivere un programma che debba visualizzare i suoi risultati in parole (ad esempio "OTTO" o "DICIASSETTE") anzichè in cifre tipo 8 o 17. Si supporrà che tutti i risultati siano noti e siano compresi tra Ø e 2Ø. È possibile impostare una tabella — la si chiamerà T$ — per tradurre le cifre in parole. Si dispone in modo che ciascun elemento contenga il nome del proprio indice, cosicchè T$(Ø) = "ZERO", T$(1) = "UNO" e così via fino a T$(2Ø) = "VENTI". Quindi per visualizzare qualsiasi numero X, basta immettere

```
PRINT T$(X)
```

Per esempio se X=8, il comando visualizza T$(8) che è la stringa "OTTO".
Naturalmente occorre fare un certo lavoro all'inizio del programma per ottenere l'impostazione della tabella. È possibile sempre scrivere una lunga lista di 21 comandi tipo:

```
T$(Ø)="ZERO"
T$(1)="UNO"
T$(2)="DUE"
...
T$(2Ø)="VENTI"
```

ma è molto meno fastidioso inserire i nomi dei numeri in un'istruzione DATA e leggerli (READ) con un'iterazione FOR. Il programma dovrebbe iniziare come segue:

```
1Ø DIM T$(2Ø)
2Ø FOR J=Ø TO 2Ø
3Ø READ T$(J)
4Ø NEXT J
5Ø DATA ZERO,UNO,DUE,TRE,QUATTRO
   CINQUE
6Ø DATA SEI,SETTE,OTTO,NOVE,DIECI
7Ø DATA UNDICI,DODICI,TREDICI,
   QUATTORDICI

8Ø DATA QUINDICI,SEDICI,DICIASSETTE
9Ø DATA DICIOTTO,DICIANNOVE,VENTI
```

Useremo ora la matrice T$ per visualizzare una tabellina per moltiplicazioni in parole. Un semplice programma che forma il punto d'inizio del nostro disegno è:

```
1ØØ FOR J=Ø TO 1Ø
11Ø PRINT "2★";J;"=";2★J
12Ø NEXT J
```

Ora modifichiamo il comando PRINT facendogli visualizzare l'appropriata entrata di tabella invece di ciascun numero.

```
"2"    diventa T$(2)
"J"    diventa T$(J)
"2★J"  diventa T$(2★J)
```

Si ottiene

```
100 FOR J= 0 TO 10
110 PRINT T$(2);"★"; T$(J);"=";
    T$(2★J)
120 NEXT J
```

Se si impostano queste istruzioni dopo quelle contrassegnate da 10 a 90, si può provare il programma.
Una proprietà base di una matrice è che se si conosce l'indice di un elemento, è possibile scegliere l'elemento ed estrarlo immediatamente. Talvolta è possibile procedere in un altro modo: si sa il valore dell'elemento e si vuole trovare dove lo si incontra (se mai) nella matrice. Quest'operazione è più difficile dato che occorre far sì che il computer cerchi nella matrice, entrata per entrata, fino a che non trova quella che corrisponde all'elemento o al limite non raggiunge la fine della matrice.
Facciamo un semplice esempio. Si vuole scrivere un programma che immette due numeri, li somma e visualizza la loro somma, ma comunica con l'utente interamente con *parole*. Per esempio, un tipico dialogo potrebbe essere

```
DATI DUE NUMERI
?OTTO,CINQUE
LA SOMMA È TREDICI
```

Entrambe le parole devono essere convertite in numeri prima che vengano sommate. La conversione da parole a numeri si verifica due volte ed è una chiara indicazione per la subroutine. La specifica e il codice per la subroutine potrebbero essere scritti abbastanza facilmente. Esse sono:

Specifiche per la subroutine

Scopo: Convertire una parola in un numero da 0 a 20

Numeri di riga: 1000-1060

Parametri: Input: A1$ Parola da convertire
Output B1: Valore del numero

Locali: JJ

Riferimento esterno: T$(0-20):
Nomi dei numeri

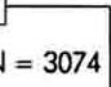
```
1000 REM CONVERTI PAROLA A1$ IN
     NUMERO B1
1010 FOR JJ=0 TO 20
1020 IF A1$ = T$(JJ) THEN 1050
1030 NEXT JJ
1040 PRINT "NON TROVATA
     ENTRATA": STOP
1050 B1 = JJ
1060 RETURN
```


Notare che la subroutine imposta un'iterazione FOR per cercare nella lista T$. Essa abbina la parola data in A1$ con T$(0), T$(1) e così via. Quando trova una corrispondente entrata esce dall'iterazione saltando al comando 1050. Se cerca nella lista e non trova un abbinamento esatto, stampa una segnalazione e si ferma.
Il programma principale è molto lineare. Eccolo, con qualche commento extra:

```
 10 DIM T$(20)
 20 FOR J=0 TO 20
 30 READ T$(J)
 40 NEXT J
 50 DATA ZERO,UNO,DUE,TRE,
    QUATTRO,CINQUE
 60 DATA SEI,SETTE,OTTO,NOVE,DIECI
 70 DATA UNDICI,DODICI,TREDICI,
    QUATTORDICI
 80 DATA QUINDICI,SEDICI,DICIASSETTE
 90 DATA DICIOTTO,DICIANNOVE,VENTI
100 PRINT"INDICA DUE NUMERI"
110 INPUT X$,Y$
120 REM IMPOSTA PARAMETRI E
    RICHIAMA SUBROUTINE PER
    CONVERTIRE X$ IN X
125 A1$=X$: GOSUB 1000:X=B1
130 REM COME PER Y
135 A1$=Y$: GOSUB 1000:Y=B1
140 Z=X+Y: REM SOMMA I DUE NUMERI
150 IF Z >20 THEN PRINT"RISULTATO
    NON IN LISTA":STOP
160 PRINT"SOMMA=";T$(Z)
170 STOP
```

Vale la pena di notare che abbiamo mantenuto tutti i dettagli di ciascun richiamo di subroutine su una riga. Ciò comprende la messa a punto del parametro di input, del comando effettivo di richiamo e dell'estrazione del risultato dal parametro di output.
Il comando 150 è chiaro in quanto il programma non può visualizzare qualsiasi numero superiore a 20. Se il comando non esistesse e l'utente battesse ad esempio, DODICI e QUINDICI, la macchina tenterebbe di accedere a T$(27). Questo elemento non esiste e visualizzerebbe:

```
? BAD SUBSCRIPT
  ERROR IN 160
```

nella nostra versione la macchina non dà alcuna risposta esatta, ma almeno il commento è un poco più informativo.

ESPERIMENTO 20·2

a) Modificare il programma nell'ultima sezione in modo che lavori con numeri ROMANI fino a XL(40)
b) Le istruzioni DATA di un programma contengono 20 nomi e numeri telefonici, disposti come segue:

```
DATA MAXWELL, 3398123
DATA BOHR, 558
DATA EINSTEIN, 4073189
DATA VON NEUMANN, 777000
DATA NEWTON, 3074
DATA ZUSE, 222
DATA PLANCK, 1237543
DATA BOYLE, 146543
DATA BABBAGE, 03474
DATA LAPLACE, 5674
DATA PTOLEMY, 54863
DATA ARISTOTELE, 66543
DATA MCCARTHY, 47
DATA DIJKSTRA, 645
DATA BERZELIUS, 777
DATA CHARLES, 5543
DATA MENDELEEV, 645634
DATA TSIOLKOVSKY, 645332
DATA ARCHIMEDES, 2
DATA HOYLE, 21352
```

Disegnare e scrivere un programma che inviti l'utente a battere un nome, quindi a cercare il nome nell'elenco e visualizzare il corrispondente numero telefonico, se lo trova. Se non lo trova, il programma dovrebbe visualizzare un adatto messaggio. Due tipiche passate potrebbero essere:

```
NOME? NEWTON

TELEFONO NEWTON = 3074
```

(NEWTON e FREUD: battuto dall'utente)

```
NOME? FREUD

FREUD NON HA
TELEFONO
```

Esperimento 20.2 completato	

Il quiz di autotest per questa unità è intitolato UNIT20QUIZ.

UNITA':21

Funzioni stringa	PAGINA 203
La funzione stringa LEN	203
La funzione stringa MID$	203
Estrazione di cognomi	203
Uso di MID$ per correggere una stringa	205
ESPERIMENTO 21.1	206
LEFT$ e RIGHT$	206
Posizionamento del cursore	206
Permutazioni - n!	206
Rimozione di lettere da una stringa	208
Conversione di stringhe in numeri - VAL	210
Conversione di numeri in stringhe - STR$	212
Arrotondamento	212
ESPERIMENTO 21.2	214
Come evitare che le parole superino la capacità dello schermo	214
ESPERIMENTO 21.3	216

LE FUNZIONI STRINGA

La manipolazione delle stringhe è una caratteristica essenziale del linguaggio BASIC che gli conferisce i poteri di risolvere i problemi pressochè in qualsiasi settore della vita quotidiana. Finora comunque i programmi che abbiamo considerato, hanno tutti interpretato le stringhe come oggetti completi e indivisibili. Ogni stringa è stata memorizzata, spostata e visualizzata esattamente nella stessa forma in cui era stata originariamente immessa nel **64**. In questa unità osserveremo qualche funzione speciale che consente di suddividere le stringhe in sezioni più piccole e anche in singoli caratteri. Queste funzioni aiuteranno a risolvere tutti i tipi di problemi che risulterebbero altrimenti difficili o impossibili da superare. Per esempio, saremo in grado di estrarre i cognomi da cognome e nome di più persone e impareremo a fare in modo che il **64** visualizzi lunghe frasi senza spezzare le parole.
Le funzioni coinvolte sono dette "funzioni stringa" in quanto usano o generano stringhe anzichè numeri. Qualsiasi funzione che come risultato genera una stringa ha un $ come ultimo carattere del suo nome, ad esempio MID$, SFR$ e così via.

La funzione stringa "LEN"

Le funzioni stringa sono costruite in BASIC in modo da poterle usare direttamente anche senza includerle in un programma. Proviamone qualcuna. Accendere la macchina e battere le seguenti righe, terminando ciascuna riga con il tasto RETURN

```
PRINT LEN("VIC")
PRINT LEN ("COMMODORE")
Z$="STRING"
PRINT LEN(Z$)
```

In ciascun caso il **64** visualizza la lunghezza (LENgth) della stringa coinvolta. In generale, la funzione LEN indica il numero di caratteri nella stringa argomento. In questo contesto "argomento" è una parola tecnica che significa l'oggetto sul quale una funzione opera.
Notare il modo in cui LEN è scritto

LEN (stringa argomento)

L'argomento deve essere racchiuso fra parentesi. Può essere una stringa esplicita racchiusa tra virgolette o il nome di una variabile stringa o qualsiasi espressione che produce una stringa come risultato. La funzione LEN produce un *numero* cosicchè l'intera costruzione può essere usata ovunque occorra un numero. Per esempio si potrebbe vedere

```
        X=LEN(Q$)
oppure  FORJ=1 TO LEN (P$)
oppure  PRINT LEN ("COMPRA" + S$ +
        "PANE")
```

La funzione stringa "MID$"

Un'altra funzione vitale è MID$. Questa funzione sceglie una *porzione* di qualsiasi stringa come suo argomento. Battere il comando

```
PRINT MID$ ("ABCDEFG",2,4)
```

Il risultato mostra come funziona MID$. In questo caso esso visualizza una stringa di quattro caratteri iniziando con il secondo carattere di "ABCDEFG".
In termini formali, la funzione MID$ prende tre argomenti che sono separati da virgole e racchiusi tra parentesi. Gli argomenti sono i seguenti:
Il primo è la stringa da usare.
Il secondo è un numero che specifica la posizione del primo carattere nel risultato. Il terzo è un altro numero che dà la lunghezza del risultato.
Come ci si potrebbe aspettare, gli argomenti possono essere variabili del tipo appropriato. La lunghezza del risultato può essere qualsiasi, da 0 (detta la stringa "nulla") fino all'intera lunghezza del primo argomento. In pratica, è spesso di un carattere.
Ecco un semplice programma per immettere una parola e visualizzarla alla rovescia. Studiarlo accuratamente e notare come vengono usate le funzioni LEN e MID$:

```
10 INPUT "BATTI UNA PAROLA"; X$
20 PRINT "PAROLA ALLA ROVESCIA="
30 FOR J = LEN(X$) TO 1 STEP -1
40 PRINT MID$(X$,J,1);
50 NEXT J
60 STOP
```

Impostare il programma e controllare il risultato; provare con parole di 1, 2 o più caratteri.

Estrazione dei cognomi

Passiamo ora ad estrarre porzioni di stringhe più ampie. Se si chiede a qualcuno di battere il proprio cognome e nome, si potrebbero avere risposte di questo genere:

```
       J.P. JONES
oppure JANET BLOGGS
oppure GEORGEP O'HAGAN
oppure ALFRED HENRY FFOUKES-SMYTHE
```

Se si vuole estrarre il cognome da una tale stringa, non è bene iniziare dal principio in quanto il computer non può individuare un cognome da un secondo nome o addirittura da una stringa di iniziali. In ogni caso il cognome viene sempre per ultimo e ciò suggerisce un modo per individuarlo. Si esamini ciascun carattere iniziando dal termine della stringa, fino a che non si arriva ad uno che non può far parte di un cognome. La successiva posizione alla destra deve essere il punto in cui inizia il cognome. Se ogni carattere nella stringa fa parte del cognome, il suo proprietario viene chiaramente da un paese tipo Afghanistan, dove la gente ha soltanto un nome.

Osservare questa stringa che ha le posizioni dei caratteri contrassegnati:

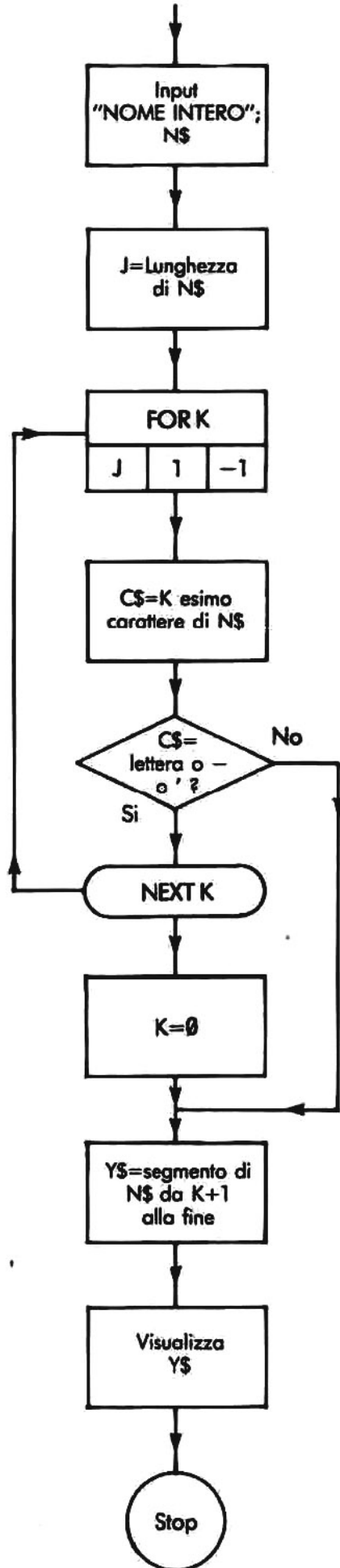
J	.	P	.	J	O	N	E	S
1	2	3	4	5	6	7	8	9

Se si inizia la ricerca da destra, il primo carattere che s'incontra che non può far parte di un cognome è il punto nella posizione 4. Il cognome pertanto inizia alla posizione 5 e può essere estratto da una funzione MID$:

MID$ (N$,5,5) (dove N$="J.P.JONES")
che dà "JONES".

In generale, se la lunghezza dell'intera stringa è J e la posizione del primo carattere che non fa parte del cognome è K, il cognome stesso inizierà al carattere (K+1) e la sua lunghezza sarà (J—K)

Quali simboli può comprendere un cognome? Gli esempi suggeriscono che lettere, trattini e apostrofi sono i soli caratteri che occorre aspettarsi.

Così sono state raccolte sufficienti idee per schizzare uno schema di flusso. Esso apparirà come segue:

Glossario

N$: Stringa col nome intero
J: Lunghezza di N$
K: È usato per analizzare all'indietro la stringa
Y$: Risultato: cognome in N$

Ora possiamo provare l'algoritmo in un breve programma e cioè:

```
 10 INPUT "NOME E COGNOME"; N$
 20 J=LEN(N$)
 30 FOR K=J TO 1 STEP -1
 40 C$=MID$(N$,K,1)
 50 IF NOT(C$ >="A" AND C$ <="Z"
    OR C$="-" OR C$="'") THEN 80
 60 NEXT K
 70 K=0
 80 Y$=MID$(N$,K+1,J-K)
 90 PRINT Y$
100 GOTO 10
```

Commenti: C$ è usato per contenere il carattere K-esimo dell'intero nome. Esso fa parte di un cognome se:

(a) è una lettera (e cioè risiede nel campo da A a Z),
oppure (b) è un trattino,
oppure (c) è un apostrofo.

L'istruzione condizionale salta a 80 se C$ *non* fa parte di un cognome.
La riga 70 viene eseguita soltanto per coloro che hanno un solo nome. Quando il comando FOR termina, la variabile controllata (K, nell'esempio), è "indefinita" (il che significa che può avere qualsiasi valore) e non necessariamente 0. Pertanto K deve essere impostato in modo che la riga 80 possa essere eseguita.
Il comando nella riga 80 estrae il cognome e lo inserisce in Y$.
Quando questo programma viene battuto, sembra funzionare correttamente su tutti gli esempi forniti. Il programma soddisfa una funzione utile e generica cosicchè lo trasformeremo in una *subroutine* con le seguenti specifiche e istruzioni. Notare il cambiamento nei nomi delle variabili:

Specifiche della subroutine

Scopo: Estrarre un cognome da un nome e cognome

Righe: Da 4100 a 4180

Parametro di input: N1$ contiene un cognome e nome

Parametro di output: Y1$ fornisce il cognome

Variabili locali: JJ,KK,CC$


```
4100 REM ESTRAI COGNOME DA N1$
     E INDICALO IN Y1$
4110 JJ=LEN (N1$)
4120 FOR KK=JJ TO 1 STEP -1
4130 CC$=MID$(N1$,KK,1)
4140 IF NOT (CC$ >="A" AND CC$ <=
        "Z" OR CC$="-" OR CC$="'")
     THEN 4170
4150 NEXT KK
4160 KK=0
4170 Y1$=MID$(N1$,KK+1,JJ-KK)
4180 RETURN
```

Un programma di gestione per provare questa subroutine potrebbe essere:

```
10 INPUT"NOME E COGNOME
   PREGO"; N1$
20 GOSUB 4100
30 PRINT "COGNOME="; Y1$
40 GOTO 10
```

USO DI MID$ PER CORREGGERE UNA STRINGA

Un'osservazione finale sulla funzione MID$: non è possibile usarla sul lato sinistro di un comando di assegnazione. Per esempio, se si vuole cambiare il quarto carattere della stringa X$ in una "U", *non è possibile* scrivere:

MID$(X$,4,1) = "U" ← (non è BASIC)

In ogni caso è possibile effettuare la stessa cosa suddividendo la stringa in tre porzioni e ricombinandola con l'operazione +:

X$=MID$(X$,1,3)+"U"+MID$(X$,5,LEN(X$)-4)

Primi 3 caratteri di X$

Fine di X$ per i caratteri da 5 in avanti

ESPERIMENTO 21·1

Questo esperimento è in tre parti:

(a) Scrivere un programma che immette una stringa dalla tastiera e la visualizza, dopo aver per prima cosa cambiato ogni "E" in una "O". Il risultato potrebbe essere:

> Quosta idoa ó stata prosa da un programma TV prosontato da Miko Bongiorno.

(b) È possibile applicare la funzione MID$ alla variabile "time" TI$ in modo da estrarre le ore, i minuti e i secondi (sotto forma di stringa) separati ciascuno da una barra. Scrivere un breve programma che visualizza l'ora corrente così:

13/23/57

(c) La subroutine per l'estrazione del cognome possiede un errore che non abbiamo trovato nelle prove originali: se qualcuno batte un punto o uno spazio dopo il cognome, la subroutine dà una stringa nulla. Disegnare un'adatta modifica per la subroutine.

Esperimento 21.1 completato	

LEFT$ E RIGHT$

Due funzioni stringa che sono spesso utili sono LEFT$ e RIGHT$. Come si può dedurre dal nome, LEFT$ estrae il lato sinistro di una stringa e RIGHT$ il lato destro. Ciascuna funzione ha due argomenti; il primo (come MID$) è la stringa da dividere e il secondo è la lunghezza del risultato. Pertanto:

```
  PRINT LEFT$("ABCDEFG",3) dà ABC
e PRINT RIGHT$("ABCDEFG",2) dà FG
```

Si sarà notato che nessuna di queste due funzioni ottiene nulla che non potrebbe essere fatto con MID$ ma esse sono tuttavia talvolta molto più utili e comode da usare.

POSIZIONAMENTO DEL CURSORE

Una particolare applicazione di LEFT$ è per posizionare il cursore in qualsiasi punto dello schermo. Inizieremo impostando due variabili stringa:

Y$ come "HOME" (posizione di partenza) seguito da numerosi "caratteri di spostamento verso il basso del cursore"

X$ come numerosi caratteri di "cursore a sinistra":

```
10 Y$="[CLR HOME][CRSR ↓]<22 volte>[CRSR ↓]"
20 X$="[CRSR ←]<39volte>[CRSR ←]"
```

Per spostare il cursore ad una posizione Y righe verso il basso a partire dalla parte superiore dello schermo, potremmo stampare (PRINT) i primi (Y+1) caratteri di Y$: un "home" (posizione di partenza) e Y volte cursore verso il basso. Analogamente spostiamo X posti orizzontalmente stampando (PRINT) i primi X caratteri di X$. Questi possono essere combinati in una singola istruzione:

```
100 PRINT LEFT$(Y$,Y+1); LEFT$(X$,X);
```

PERMUTAZIONI — n!

Il successivo esempio riguarda le permutazioni. Le permutazioni sono utili per risolvere problemi e trovare anagrammi di parole e in un contesto più serio essi giocano un ruolo importante nelle statistiche e nel progetto di esperimenti scientifici. La sezione contiene un po' di matematica, ma se si trova la matematica difficile è possibile usare il programma di permutazione senza leggere la spiegazione. Se l'intero concetto è troppo ostico, saltare questa parte. Le permutazioni non sono una parte essenziale della programmazione in BASIC.

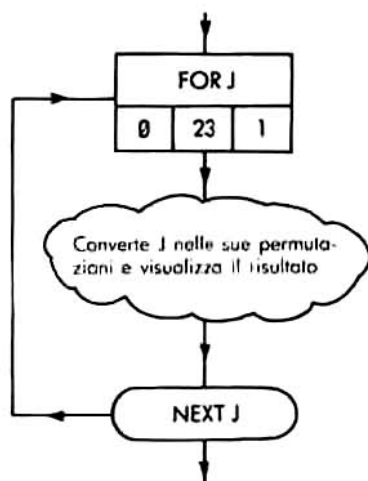
Una *permutazione* è un particolare ordine di disposizione di una serie di oggetti o di eventi. Per esempio, l'ordine in cui viene fatto suonare uno scampanio di otto campanelli è una permutazione e così l'ordine in cui i cavalli in una gara

tagliano il traguardo. Quante permutazioni è possibile ottenere? Ciò dipende dal numero degli oggetti. In una gara con un solo cavallo può esserci soltanto un vincitore. Se ci sono due cavalli denominati A e B uno solo può vincere, ma l'altro deve essere per forza secondo: ci sono quindi due permutazioni e cioè AB e BA. Con tre cavalli, possono esserci tre differenti vincitori e in ciascun caso uno dei due cavalli rimanenti può essere secondo. Ci sono pertanto sei permutazioni: ABC, ACB, BAC, BCA, CAB e CBA. Con 4 cavalli possono esserci 4x3x2 ossia 24 differenti risultati. La tabella mostra il modo in cui si producono queste cifre.

N. di oggetti	*N. di permutazioni*
1	1
2	2 = 2
3	3x2 = 6
4	4x3x2 = 24
5	5x4x3x2 = 120
6	6x5x4x3x2 = 720
7	7x6x5x4x3x2 = 5040
8	8x7x6x5x4x3x2 = 40320

Come si può vedere il numero di permutazioni cresce molto rapidamente e supera rapidamente i tre milioni per 10 oggetti. Il numero di permutazioni di "n" oggetti è denominato "fattoriale di n ", una frase che matematicamente si scrive talvolta nella forma "n!", per indicare il prodotto di tutti i numeri da 1 a n. Risolveremo ora un programma che legge qualsiasi stringa e ne visualizza tutte le permutazioni. Per esempio, se l'input è "TEA" l'output dovrebbe comprendere TEA, TAE, ATE, AET, EAT e ETA.

Si supponga che la stringa iniziale sia lunga n lettere. Sappiamo che ci saranno n! (fattoriale di n) diverse permutazioni, ma come è possibile far sì che il computer le elabori senza ripetersi o mancandone qualcuna?

Un modo per affrontare questo problema consiste nell'inventare un metodo per convertire i numeri 0,1,2,3... in permutazioni in modo che ciascuna sia diversa da tutte le altre. Quindi se n è — ad esempio 4 — (una stringa di quattro lettere) è possibile produrre tutte le 24 permutazioni di quattro lettere convertendo ciascuno dei numeri da 0 a 23 nella corrispondente permutazione e visualizzando i risultati. Lo schema di flusso per tale programma sarebbe:

In questo schema di flusso possiamo chiamare J come "numero di permutazioni". Abbiamo ancora bisogno di trovare un modo per convertire il numero di permutazioni nella corrispondente permutazione di lettere. Questo problema non è facile, ma potremmo ottenere qualche indicazione osservando le risposte in alcuni semplici casi. Si supponga che gli oggetti da permutare siano le lettera A B C e così via.

1 Oggetto: 1 Permutazione

A

2 Oggetti: 2 Permutazioni

AB
BA

3 Oggetti: 6 Permutazioni

ABC ACB
BAC BCA
CAB CBA

4 Oggetti: 24 Permutazioni

ABCD ABDC ACBD ACDB ADBC ADCB
BACD BADC BCAD BCDA BDAC BDCA
CABD CADB CBAD CBDA CDAB CDBA
DABC DACB DBAC DBCA DCAB DCBA

Si vedrà che se ciascuna permutazione viene presa come una "parola" tutte le parole della stessa dimensione sono scritte nell'ordine del dizionario.

Quando si studiano queste liste, emerge un profilo ben definito. Si supponga di dividere i membri in ciascuna serie secondo le rispettive prime lettere e quindi le permutazioni di tre lettere in tre gruppi di due ciascuno:

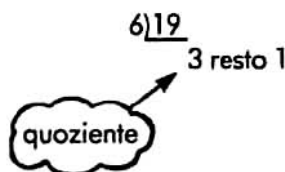
ABC BAC CAB
ACB BCA CBA

In ciascun gruppo la lettera iniziale è seguita da tutte le permutazioni delle altre due. Così A è seguito e BC e CB "subpermutazioni".

È possibile vedere questo profilo comparire anche nelle permutazioni di quattro lettere. Ci sono quattro gruppi, ciascuno con sei membri. Ciascuna lettera iniziale è seguita da sei subpermutazioni delle altre tre.

Per qualsiasi permutazione possiamo definire un "numero di gruppo" e un "numero di subpermutazione". Il numero di gruppo indicherà la prima lettera (secondo il codice A=0, B=1, C=2 e così via) e il numero di subpermutazione sarà la posizione all'interno del gruppo (sempre iniziando da 0). Per esempio si consideri la permutazione BCDA. Questa ha un numero di gruppo di 1 (in quanto inizia con un B) e il numero di subpermutazione è 3, come è possibile controllare dalla tabella che precede.

C'è ora un importante suggerimento circa il metodo di convertire qualsiasi numero di permutazione nella corrispondente permutazione. Troviamo per prima cosa il numero di gruppo, che definisce la prima lettera; quindi troviamo il numero di subpermutazione e ricaviamo la corrispondente permutazione dalle lettere rimanenti!

Per trovare il numero di gruppo e di supermutazione tutto ciò che occorre fare è di dividere il numero di permutazioni per la dimensione del gruppo. Il quoziente dà il numero di gruppo e pertanto la prima lettera e il resto specifica il numero di subpermutazione.
Per dare un esempio, si consideri il numero di permutazione 19 di quattro lettere.

La corrispondente permutazione inizia con D (D=3) ed è seguita dal numero di subpermutazione 1 delle lettere A B C. La subpermutazione può essere ricavata esattamente con lo stesso processo della permutazione. Ci sono soltanto 2 importanti modifiche:

1) La lettera usata all'inizio della permutazione principale deve essere rimossa dalla lista di lettere in modo che non venga utilizzata di nuovo

2) La dimensione del gruppo deve essere aggiustata (ad esempio da 6 a 2 o da 2 a 1).

Qui c'è uno specifico esempio, che prende la permutazione 9 di quattro lettere. Iniziamo contrassegnando le lettere A=Ø, B=1, C=2, D=3.

1) Dividiamo 9 per 6 ottenendo il quoziente = 1, resto = 3. La prima lettera della permutazione è pertanto B. Rimuoviamo la B dalla lista delle lettere e ricontrassegnamo le altre: A=Ø, C=1, D=2.

2) Troviamo ora la subpermutazione 3 dalle lettere A, C e D. La dimensione di gruppo è 2. Dividendo 3 per 2 otteniamo il quoziente = 1, il resto = 1. La successiva lettera della permutazione è pertanto C. Rimuoviamo la C dalla lista e ricontrassegnamo le altre lettere A=Ø, D=1.

3) Troviamo ora la subpermutazione 1 dalle lettere A e D. La dimensione del gruppo è 1. Dividendo 1 per 1, si ottiene il quoziente 1, resto = Ø. La successiva lettera della permutazione è D. Rimuoviamo questa lettera dalla lista. Ciò lascia una sola lettera, una A contrassegnata Ø.

4) Troviamo infine la subpermutazione Ø dalla lettera A. Ovviamente è la A, ma possiamo ancora usare lo stesso processo usato in precedenza: la dimensione del gruppo è 1 e Ødiviso 1 dà come quoziente = Ø, come resto = Ø. Come previsto, la lettera finale è A e la permutazione nel suo complesso è BCDA.

Per verificare la comprensione di questo processo, provare a convertire qualche numero tra Ø e 23 e assicurarsi che le risposte risultino identiche alla tabella che precede. (Ricordarsi che la prima permutazione ABCD corrisponde a Ø non a 1).
Ora convertiremo il metodo in un programma. Le lettere da permutare non sono necessariamente ABCD ma possono essere qualsiasi cosa l'utente batte. Analogamente la lunghezza della stringa è arbitraria quantunque possiamo aspettarci che il programma giri per un tempo lunghissimo se ci sono più di 6 o sette lettere. Innanzitutto dobbiamo gestire un "lotto" di lettere e assicurarci che siano selezionate correttamente. Il modo più semplice per far ciò consiste nell'inserirle in una stringa (ad esempio Y$). Dato che sono quindi numerate automaticamente, la lettera con la label "notazionale" Q, può essere scelta nella forma:

```
MID$(Y$,Q + 1,1)
```

Il "+1" deve essere incluso in quanto lo schema di numerazione automatica inizia a 1, mentre il nostro metodo produce numeri di gruppi che iniziano da Ø.

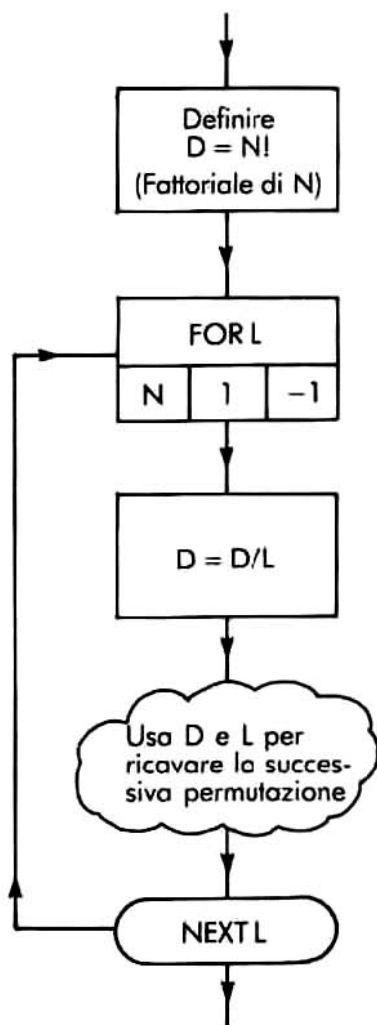
RIMOZIONE DELLE LETTERE DA UNA STRINGA

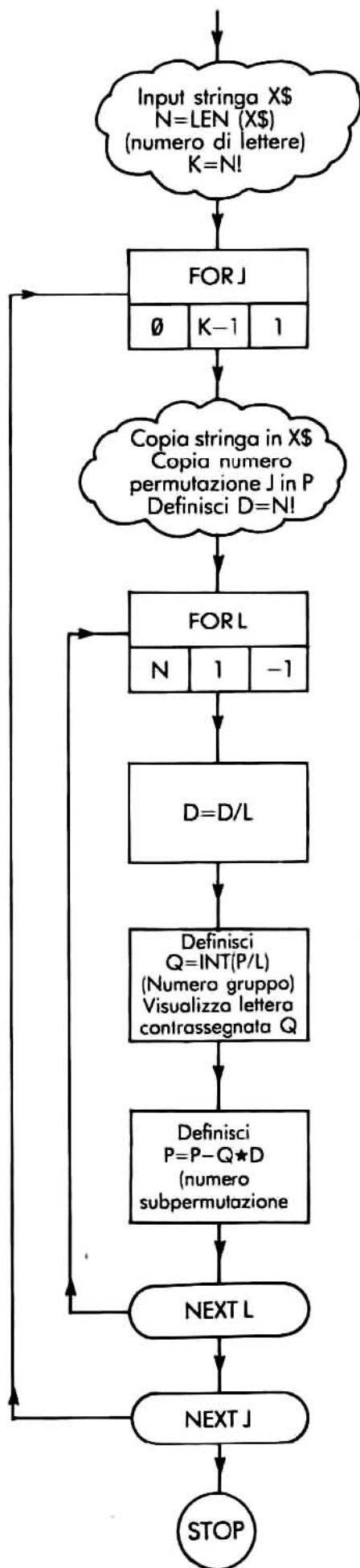
Una volta che una lettera è stata usata, deve essere rimossa dalla stringa. Le altre saranno quindi spostate verso l'alto automaticamente il che equivale a ricontrassegnarle. Estrarre una lettera da una stringa è abbastanza facile: si concatenano (si uniscono) la porzione della stringa alla sinistra della lettera utilizzata e la porzione alla destra.

Risultato: HORRBLE

Se il numero di label della lettera risultato è Q, la porzione di sinistra avrà Q lettere e la porzione di destra (LEN(Y$) −Q1). (Ricordarsi che le label vanno da Ø a Q). Il comando richiesto è:

```
Y$=LEFT$(Y$,Q) + RIGHT$(Y$,LEN(Y$)
−Q−1)
```

Secondo, il numero di lettere di ciascuna subpermutazione diventa 4 3 2 1 e la corrispondente dimensione del gruppo . . . 6 2 1 1. Questi ultimi sono i valori di n! per diversi valori di n e possono essere prodotti da un programma tipo questo:

In questo schema di flusso L e D prenderanno le corrette sequenze di valori. Per esempio se N=6, L diventa 6,5,4,3,2,1 e D (nel momento in cui viene usato) sarà 120,24,6,2,1 e 1.
Possiamo ora riunire tutte queste idee in un programma. Il processo per convertire una stringa in una permutazione gradualmente la distrugge cosicchè dobbiamo tenere una copia base dell'originale e sostituire la "copia di lavoro" per ciascuna permutazione separata. Lo schema di flusso e il programma sono:

Il Glossario è
X$: Stringa da permutare
N: Lunghezza della stringa da permutare
K: Fattoriale di N (calcolato nella riga 30)
J: Numero di permutazione
Y$: Copia di lavoro di X$
D: Dimensione corrente del gruppo
L: Numero delle lettere nella subpermutazione corrente
P: Numero di permutazione corrente
Q: Numero di gruppo della subpermutazione corrente
S: Variabile per tutti i numeri da 1 a N

Dopo questa complessa analisi, il programma risulta sorprendentemente corto. Esso è:

```
 10 INPUT X$
 20 N=LEN(X$)
 30 K=1:FOR S=1 TO N:K=K*S:NEXT S
 40 FOR J=0 TO K -1
 50 Y$=X$: D=K: P=J
 60 FOR L=N TO 1 STEP -1
 70 D=D/L
 80 Q=INT(P/D): P=P-D*Q
 90 PRINT MID$(Y$,Q+1,1)
100 Y$=LEFT(Y$,Q)+RIGHT$(Y$,LEN(Y$)
    -Q-1)
110 NEXT L
120 PRINT
130 NEXT J
140 STOP
```

Il quoziente e il resto sono calcolati nella riga 80. Ricordarsi che INT scarta qualsiasi frazione; ciò fa sì che il comando funzioni correttamente. Per esempio se P=17 e D=6, allora

Q=INT(17/6) = INT(2.8333333) = 2
P=17 – 6*2 = 17 – 12 = 5

Impostare questo programma e provarlo con i propri dati. Vedere se è possibile modificarli in modo che visualizzi più di una permutazione sulla stessa riga.

CONVERSIONE DI STRINGHE IN NUMERI - VAL

Due altre funzioni stringa aiutano a convertire stringhe in numeri e viceversa.

VAL ("una stringa")

prende una stringa di cifre decimali (eventualmente preceduta da + o – e contenente un punto decimale) e la converte nel corrispondente valore numerico.
VAL è utile per ottenere un input valido da utenti molto ingenui. Si consideri un programma che chiede di battere un numero, mediante un'istruzione del tipo

INPUT X

Se l'utente batte effettivamente qualcosa che non è un numero ad esempio "PARDON", il sistema BASIC dice semplicemente

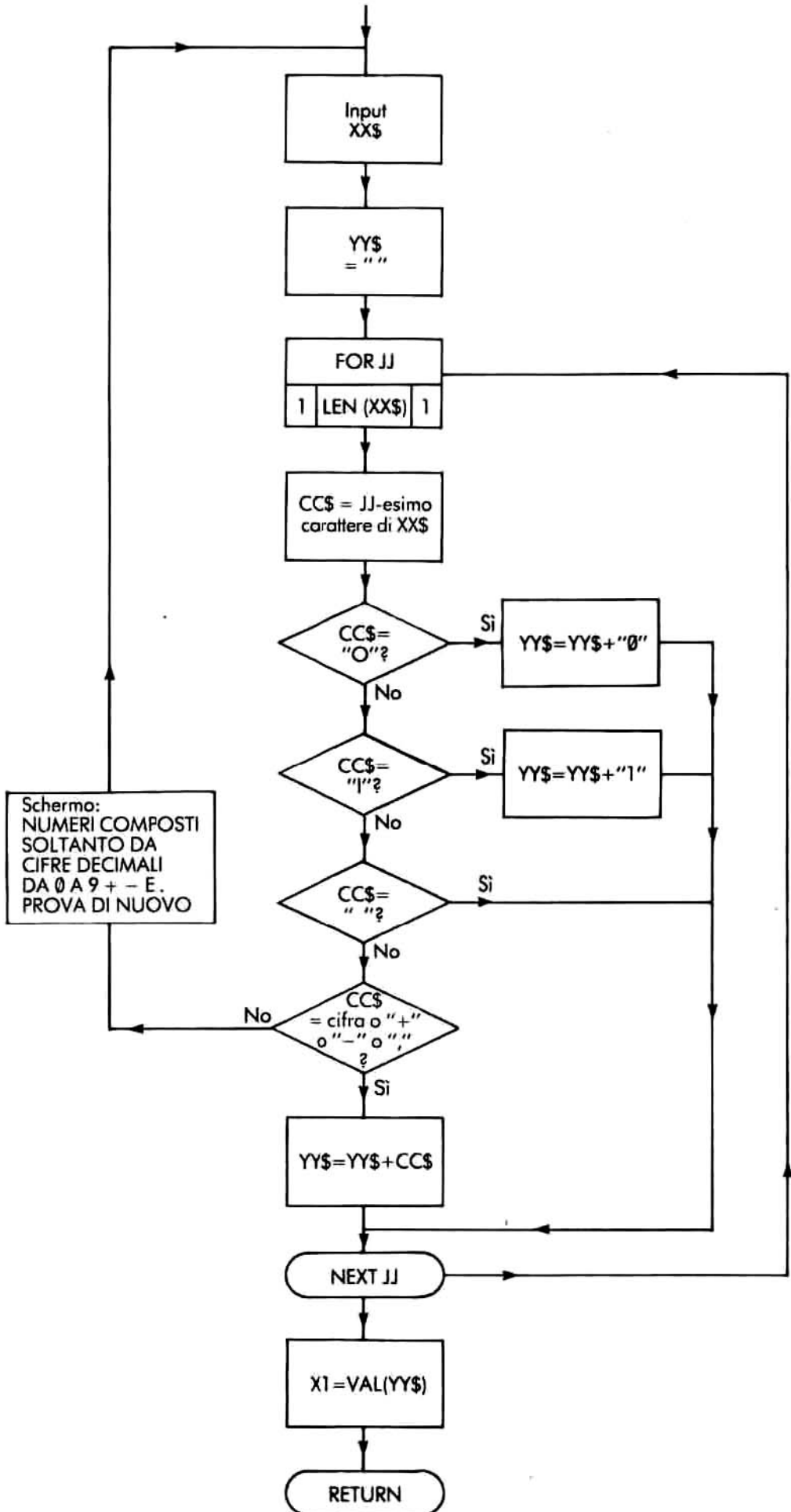
REDO FROM START

Ciò non è molto utile e l'utente può non rendersi conto di che cosa ci si aspettava da lui.
Come alternativa, tutto ciò che l'utente batte può essere letto come stringa, quindi non c'è rischio di un messaggio REDO FROM START e il programma può analizzare la risposta dell'utente, carattere per carattere emettendo adatti messaggi di errore se rileva qualche sbaglio.
Infine, se la stringa risulta composta da simboli che costituiscono un numero accettabile, il valore può essere estratto con VAL. Qui c'è una specifica, uno schema di flusso e una subroutine per input "tollerante" di numeri.

Specifiche della subroutine

Scopo: Far immettere numeri da un utente inesperto.
Tutti gli spazi sono ignorati e le lettere I e O sono interpretate 1 e 0.
Altri errori sono chiaramente spiegati.

Righe: da 4500 e 4660

Parametro di output: Il risultato è fornito in X1

Variabili locali: XX$, YY$, JJ$, CC$

Input
XX$

YY$
= " "

FOR JJ
1 | LEN (XX$) | 1

CC$ = JJ-esimo
carattere di XX$

CC$=
"O"?
Sì
YY$=YY$+"Ø"
No

CC$=
"l"?
Sì
YY$=YY$+"1"
No

CC$=
" "?
Sì
No

CC$
= cifra o "+"
o "−" o ","
?
No
Sì

Schermo:
NUMERI COMPOSTI
SOLTANTO DA
CIFRE DECIMALI
DA Ø A 9 + − E.
PROVA DI NUOVO

YY$=YY$+CC$

NEXT JJ

X1=VAL(YY$)

RETURN

```
4500 REM INPUT TOLLERANTE DI NUMERI
4510 INPUT XX$
4520 YY$=""
4530 FOR JJ = 1 TO LEN (XX$)
4540 CC$=MID$(XX$,JJ,1)
4550 IF CC$="O" THEN YY$=YY$+"0":
     GOTO 4600
4560 IF CC$="I" THEN YY$=YY$+"1":
     GOTO 4600
4570 IF CC$="" THEN 4600
4580 IF NOT(CC$<="9" AND CC$>="0"
     OR CC$="+" OR CC$="-"OR
     CC$=".") THEN 4620
4590 YY$=YY$+CC$
4600 NEXT JJ
4610 X1=VAL(YY$):RETURN
4620 PRINT "UN NUMERO È COSTITUITO
     SOLO DA"
4630 PRINT "CIFRE 0-9,"
4640 PRINT "+,-,."
4650 PRINT "PREGO TENTA ANCORA"
4660 GOTO 4510
```

CONVERSIONE DI NUMERI IN STRINGHE—STR$

STR$ (un numero) funziona all'opposto di VAL. Esso prende un numero come argomento e fornisce una stringa di simboli, gli stessi che sarebbero stati visualizzati se fosse stato usato PRINT. STR$ è una funzione preziosa per ottenere sullo schermo un tracciato nitido di numeri.

Il problema principale con il comando PRINT è che non è mai possibile sapere quale sarà il tracciato esatto. Per illustrare questo punto, impostare il seguente programma:

```
 5 PRINT"NUMERO"," ","QUADRATO"
10 FOR J=1 TO 7 STEP 0.1
20 PRINT J," ",J★J
30 NEXT J
40 STOP
```

Ricordarsi di includere le stringhe vuote (" ") nelle righe 5 e 20, altrimenti l'esempio non funziona nel modo previsto.

Eseguire questo programma lentamente tenendo abbassato il tasto CTRL. All'inizio tutto sembra andare bene. Lo schermo visualizza la tabella dei quadrati che vi si aspettava. Si ottiene.

```
1        1
1.1      1.21
1.2      1.44
```

e così via.

Per contro, l'entrata 2.8 sembra peculiare; essa dice

```
2.8        7.8399999
```

ed è seguita da una riga vuota. Si sa perfettamente che il quadrato di 2.8 è 7.84, e non 7.83999999 come compare sullo schermo.

Dopo 3.6 la tabella impazzisce. Essa indica:

```
3.6                     12.96
3.69999999
13.69
3.79999999
14.44
3.89999999
15.21
3.99999999
```

e così via.

La difficoltà è dovuta ai due effetti che interagiscono l'uno con l'altro.

Il primo problema è quello "dell'errore di troncamento". Dato che il **64** — come la maggior parte dei computer — lavora sul sistema binario, non può gestire esattamente numeri tipo 0.1. C'è sempre un piccolo errore. Nel nostro programma il valore di J inizia con 1 e cresce verso 7 mediante ripetute aggiunte di 0.1; al limite gli errori si accumulano e compaiono in risultati che sono pressochè esatti ma non abbastanza quanto ci si aspettava. Pertanto, la differenza tra

7.84 e 7.83999999

è soltanto 0.00000001, ma ciò è sufficiente per mettere scompiglio nel tracciato; il numero sembra abbastanza diverso e viene inserita forzatamente una riga vuota in quanto l'ultima cifra del numero va oltre l'ultima colonna dello schermo.

Il secondo emerge quando l'errore di troncamento interessa il valore stampato dalla J. "3.7" è trasformato in "3.6999999" e questo numero è così lungo che finisce nella parte dello schermo dove dovrebbe comparire normalmente il secondo numero. Il risultato costringe il **64** ad iniziare una nuova riga per il secondo numero nell'istruzione PRINT e così distrugge l'intero aspetto della tabella.

STR$ consente un controllo migliore sul tracciato dei numeri decimali. Esso prende un argomento numerico e produce una stringa in cifre decimali, spazi, ecc. che è la stessa che sarebbe stata visualizzata dall'istruzione PRINT. La differenza è che il risultato è *interno* e può essere manipolato e corretto prima di venire visualizzato. Per illustrare questo punto ecco un breve programma che visualizza un numero scritto alla rovescia:

```
10 INPUT "DATO UN NUMERO"; X
20 X$=STR$(X)
30 FOR J=LEN(X$) TO 1 STEP -1
40 PRINT MID$ (X$,J,1);
50 NEXT J
60 PRINT
70 GOTO 10
```

ARROTONDAMENTO

La prima tecnica che esamineremo è quella dell'*arrotondamento*. Il **64** generalmente visualizza frazioni fino a 8 cifre decimali salvo che esclude gli zeri di coda. Solitamente, 3 o 4 cifre decimali sono un'accuratezza sufficiente per i risultati. Quando un decimale è accorciato mediante arrotondamento, esso solitamente aggiunge 1

all'ultima cifra conservata se la parte scartata inizia con 5 o più. Per esempio se il valore corretto di un numero è 3.14159, il valore arrotondato (a tre cifre decimali) è 3.142. Per contro, il valore arrotondato di 2.71828 è 2.718.
La meccanica dell'arrotondamento è abbastanza lineare. Per arrotondare un numero positivo, a tre cifre, aggiungiamo 0.0005 e quindi scartiamo la quarta e le successive cifre. Questi esempi mostrano il processo al lavoro:

3.13159	2.71828
0.0005 +	0.0005 +
3.14209	2.71878
(scartare 09)	(scartare 78)
3.142	2.178

È chiaro che l'arrotondamento si libera degli errori di troncamento introdotti dal **64** (dato che 3.69999999 arrotondato a tre cifre diventa 3.700) ed evita inoltre le variazioni imbarazzanti nel numero dei caratteri visualizzati.
Il processo per stampare i numeri arrotondati a 3 cifre decimali è il seguente:

(1) Aggiunta di 0.0005

(2) Uso di STR$ per convertire in forma di stringa (ad esempio) NN$

(3) Individuazione della posizione del punto decimale (ad esempio) PP

(4) Visualizzazione della parte di sinistra di NN$ fino a 3 cifre dopo il punto decimale.

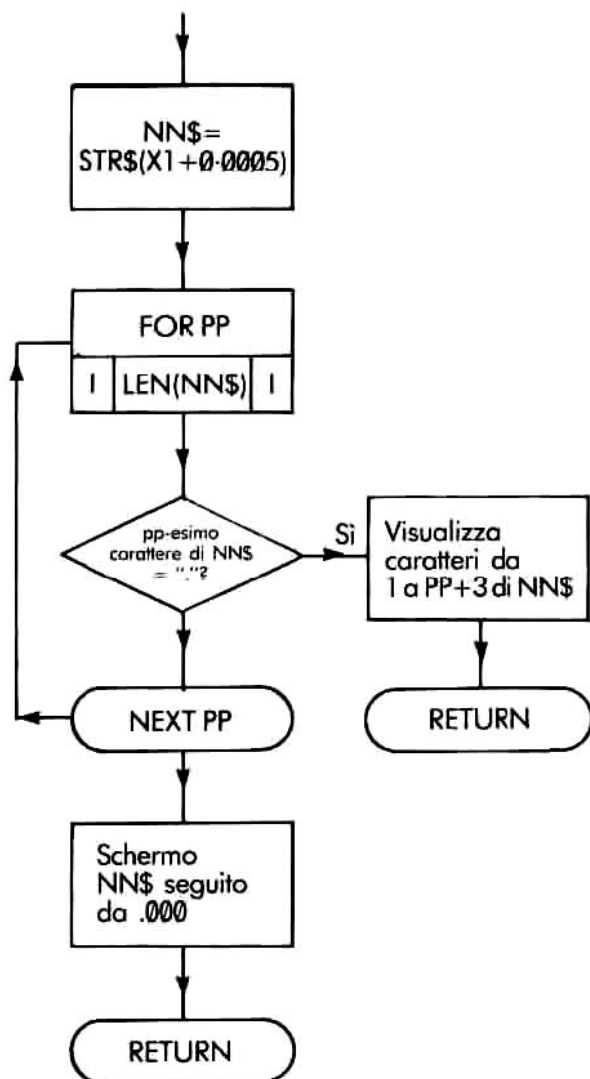
Potrebbe darsi che non ci sia punto decimale in NN$. Ciò succederebbe se il valore originale terminasse in ".9995" (ad esempio 2.9995"). Quindi NN$ comparirebbe come "3" senza punto decimale. Ciò deve essere considerato un caso speciale.
Possiamo introdurre questo algoritmo in un'utile subroutine.

Specifiche della subroutine

scopo: Visualizzare un numero positivo arrotondato a 3 cifre decimali

Righe: da 5000 a 5050

Parametro di input: X1 ha il valore di numero

Variabili locali: NN$, PP

La corrispondente codifica è:

```
5000 REM VISUALIZZA X1 A 3 CIFRE
       DECIMALI
5010 NN$=STR$(X1 + 0.0005)
5020 FOR PP=1 TO LEN(NN$)
5030 IF MID$(NN$,PP,1)="."THEN
       PRINT LEFT$(NN$,PP+3);:RETURN
5040 NEXT PP
5050 PRINT NN$;".000";:RETURN:REM
      NIENTE DECIMALI IN NN$
```

Un'adatta routine di gestione è una versione modificata del programma che ci dava tutti quei fastidi originariamente:

```
10 FOR J=1 TO 7 STEP 0.1
20 X1=J: GOSUB 5000
30 X1=J*J: GOSUB 5000
40 PRINT
50 NEXT J
60 STOP
```

Se si esegue questo programma, si vedrà che tutte le difficoltà scompaiono completamente!

(a) Modificare la subroutine della visualizzazione discusso precedentemente in modo che il programma principale possa scegliere il numero di cifre decimali da usare. Questo numero sarà fornito come *parametro* in Y1. Suggerimenti: la costante da aggiungere può essere scritta nella forma:

```
0.5*10↑-Y1
```

Provare la subroutine a fondo e usarla per visualizzare alcune nuove tabelle.

Esperimento 21.2 completato	

COME EVITARE CHE LE PAROLE SUPERINO LA CAPACITÀ DELLO SCHERMO

I personal computer che usano i televisori come schermo, hanno in genere delle limitazioni per quanto riguarda il numero di caratteri che possono visualizzare sullo schermo stesso. Ciò non è dovuto a carenza del computer ma dalla scarsità di definizione tipica del televisore domestico, che rende poco leggibili i piccoli caratteri. Il **64** consente di scrivere 40caratteri su ciascuna riga, come altri computer del suo genere e con risultati anche migliori. Se si desidera scrivere più di 40 caratteri, occorre comprare una macchina più costosa con un proprio monitor.

La larghezza di schermo sul **64** non è un inconveniente serio nella programmazione ma richiede cura per visualizzare messaggi in modo che le parole non vengano spezzate. Se si usa una serie di comandi PRINT, occorre osservare le seguenti regole:

(1) Nessuna riga deve comprendere più di 40 caratteri.

(2) Se la riga ha esattamente 40caratteri, il comando PRINT deve seguire il testo con un punto e virgola per impedire che venga introdotta forzatamente una riga vuota. Ciò in quanto un carattere nella 40esima posizione provoca sempre l'inizio di una nuova riga.

Per terminare questa unità, descriveremo una subroutine, che dispone automaticamente il testo in modo da evitare questo problema.

Si supponga di avere una stringa di parole separate da spazi. La stringa può avere qualsiasi lunghezza fino ad un massimo di 255 caratteri. Se ci limitiamo a scriverla (PRINT), essa verrà tagliata a righe di 40caratteri senza qualsiasi riguardo alle posizioni delle parole e degli spazi. Dobbiamo trovare un metodo migliore per suddividerla in righe.

Se la stringa è di 40caratteri o meno, può essere visualizzata così come è. Altrimenti dobbiamo esaminare la stringa e trovare il segmento più ampio (iniziando dal principio), che può essere visualizzato senza tagliare una parola in due. Visualizziamo quel segmento, lo rimuoviamo dalla parte anteriore della stringa e iniziamo di nuovo il processo su ciò che rimane. Per trovare il segmento più ampio, cerchiamo uno spazio che inizia al 41esimo carattere e cerchiamo all'indietro. Per vedere il perché, si consideri la stringa

FRIENDS,ROMANS,COUNTRYMEN,
LEND ME YOUR EARS

Il 41esimo carattere è la R in YOUR, così cerchiamo all'indietro fino a che arriviamo allo spazio che si trova al carattere 37.

Visualizziamo la riga di 36 caratteri

FRIENDS,ROMANS,COUNTRYMEN,LEND ME

e rimuoviamo 37 caratteri dalla parte anteriore della stringa, lasciando

che trova posto facilmente sulla riga successiva. Una specifica di subroutine, lo schema di flusso e la codifica per questo processo, sono tutti indicati qui di seguito.

Specifiche della subroutine

Scopo: Visualizzare la stringa X1 in modo che le parole non vengano spezzate su due righe

Righe: da 5500 a 5600

Parametro di input: X1$

Variabili locali: XX$, PP, QQ, RR

XX$=X1$

PP=LEN(XX$)

PP<=40?

Sì

RR=PP

Stampa i caratteri RR più a sinistra di XX$ su una riga

RETURN

No

FOR QQ

41	1	−1

QQ-esimo carattere di XX$= " "?

Sì

RR=QQ−1

Stampa RR caratteri più a sinistra di XX$ su una riga

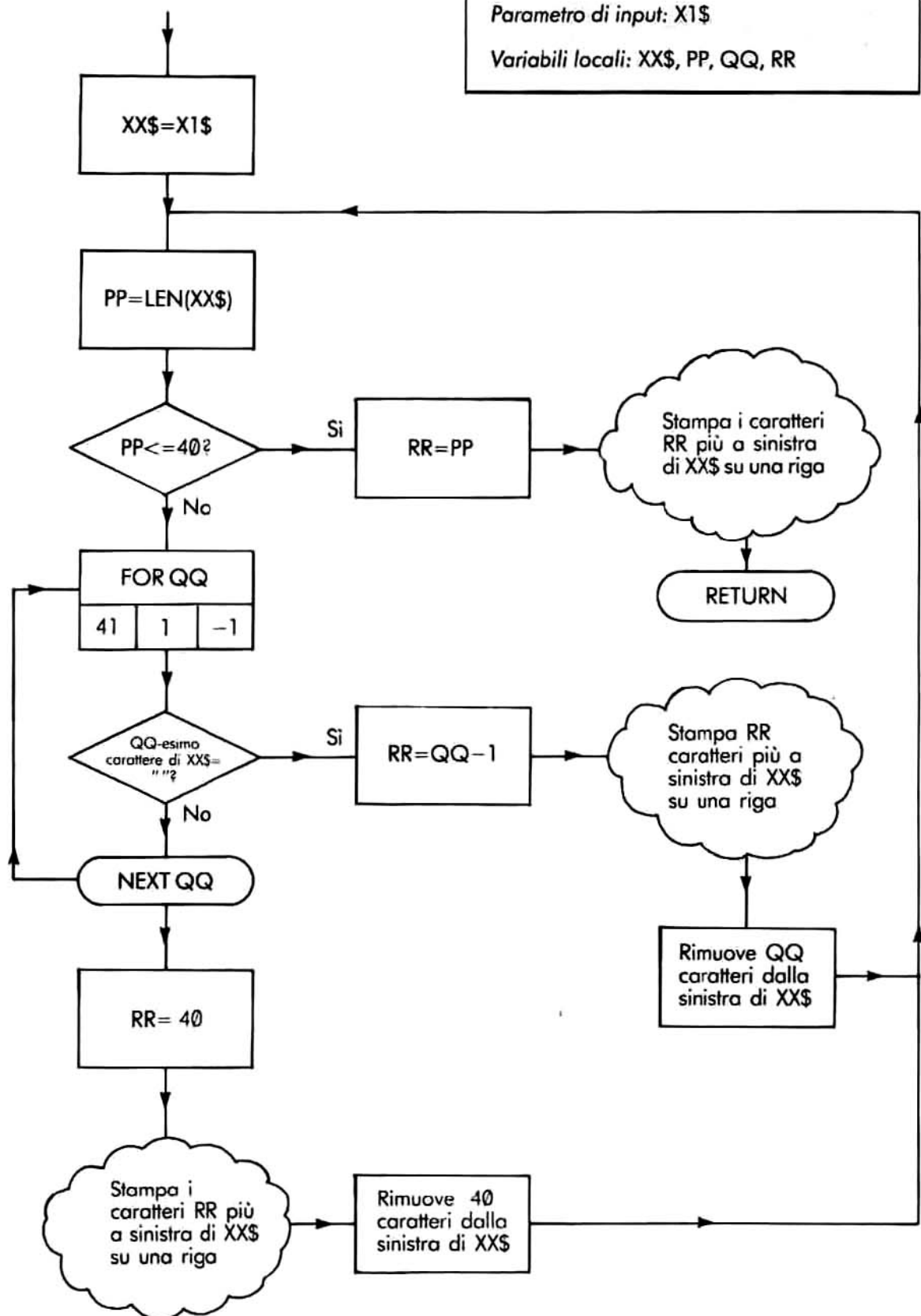
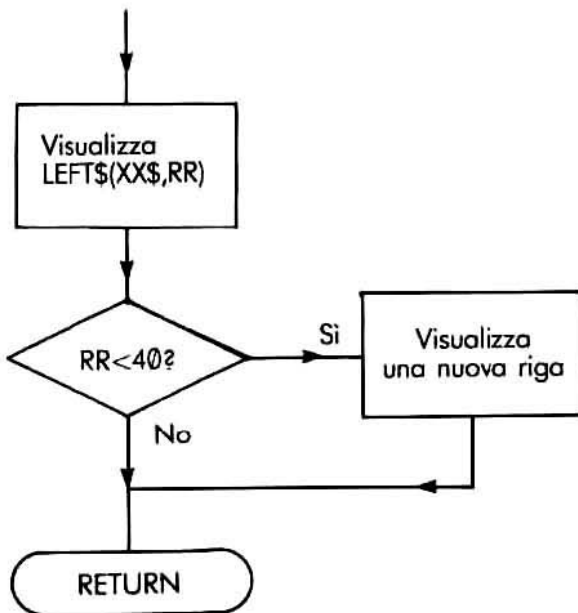
Rimuove QQ caratteri dalla sinistra di XX$

No

NEXT QQ

RR= 40

Stampa i caratteri RR più a sinistra di XX$ su una riga

Rimuove 40 caratteri dalla sinistra di XX$

(subroutine interna)

```
5500 REM VISUALIZZA X1$ SENZA
     DIVIDERE LE PAROLE
5510 XX$=X1$
5510 PP=LEN(XX$)
5530 IF PP<=40 THEN RR=PP:GOSUB
     5580:RETURN
5540 FOR QQ=41TO 1 STEP -1
5550 IF MID$(XX$,QQ,1)=" "THEN
     RR=QQ -1:GOSUB 5580:
     XX$=RIGHT$(XX$,PP-QQ):GOTO
     5520
5560 NEXT QQ
5570 RR=   :GOSUB 5580:XX$=RIGHT$
     (XX$,PP-40):GOTO 5520
5580 REM SUBROUTINE INTERNA
5590 PRINT LEFT$(XX$,RR):IF RR<40
     THEN PRINT
5600 RETURN
```

Un adatto programma di gestione per questa subroutine è:

```
10 X1$="BATTI QUALSIASI STRINGA
   LUNGA FINO A TRE RIGHE"
15 X1$=X1$+"PER PROVARE
   SUBROUTINE TRACCIATO"
20 GOSUB 5500
30 INPUT X1$:GOSUB 5500
40 GOTO 10
```

ESPERIMENTO 21·3

(a) L'utente di un programma batte una stringa che contiene un numero ed eventualmente una parola (ma non necessariamente) separati da 1 o più spazi. La stringa di input potrebbe essere

```
        3 APPLES
oppure 174 PETS
oppure   1 QUEUE
```

Scrivere un programma che estrae la parola e il numero e li visualizza nell'ordine opposto con il numero raddoppiato cioè:

```
APPLES    6
PETS    348
QUEUE     2
```

SUGGERIMENTO: Usare MID$ e VAL per estrarre i numeri.

(b) In risposta ad una domanda un utente potrebbe scrivere una stringa tipo questa:

```
IO VOGLIO 6 ARANCE 17 MELE
2 POMPELMI 157 NOCI
E 15 MELONI
```

Scrivere una subroutine che analizza tale stringa e imposta le variabili come segue:

Matrice N1$:	I *nomi* delle varie voci richieste
Matrice Q1:	Le quantità delle varie voci
Variabili X:	Numero delle diverse voci richieste

Per esempio, la frase suddetta dovrebbe dare:

```
N1$(1) = "ARANCE"      Q1(1) = 6
N1$(2) = "MELE"        Q1(2) = 17
N1$(3) = "POMPELMI"    Q1(3) = 2
N1$(4) = "NOCI"        Q1(4) = 157
N1$(5) = "MELONI"      Q1(5) = 15

              X = 5
```

La subroutine dovrebbe ignorare le parole IO, VOGLIO, VORREI, E.
SUGGERIMENTO: analizzare la stringa con un puntatore e usare MID$ per estrarre sequenze di lettere o cifre ciascuna terminante con uno spazio.

Esperimento 21.3 completato	

Il quiz di autotest per questa unità è intitolato UNIT21QUIZ64.

UNITA':22

Altri usi delle matrici-Ricerca e riordino	PAGINA 219
La "Ricerca dicotomica"	220
Esperimento 22.1	222
Metodo di ordinamento "Bubble sort"	222
Esperimento 22.2	223
Quicksort	224
Confronto dei tempi di riordino	225
La memoria del COMMODORE 64	225
Matrici bidimensionali	226
Esperimento 22.3	228

ALTRI USI DELLE MATRICI — Ricerca e riordino

Questa unità introduce ulteriormente allo studio delle matrici e del modo in cui sono usate. Si esamineranno la *ricerca* e il *riordino*, due tecniche che sono estremamente importanti per molte moderne applicazioni di computer.
L'esperimento al termine dell'Unità 20 chiederà di scrivere un programma per cercare in una lista di nomi contenuti in una matrice. Se ci sono solo pochi numeri questo processo non è difficile. Si inizia dall'alto e si lavora procedendo verso il basso, fermandosi soltanto quando si trova un esatto accoppiamento oppure quando si raggiunge il fondo della lista e si esauriscono i nomi da ricercare. Un frammento di codice che comporta tale richiesta, potrebbe essere il seguente:


```
120 FOR J= 1 TO 12
130 IF X$=A$(J) THEN 170
140 NEXT J
150 PRINT "NESSUN ABBINAMENTO"
160 STOP
170 PRINT "ABBINAMENTO IN"; J
180 STOP
```

Glossario

X$: Nome da cercare
A$(1-12): Matrice di nomi da cercare
J: Puntatore all'elemento corrente in A$

Nel discutere i metodi di ricerca il nome (o numero) che viene cercato è detto "bersaglio" e l'atto di abbinarlo a fronte di un'entrata nella lista è detto un "confronto". Nell'esempio, il bersaglio è in X$ e il confronto si verifica nella riga 130.
In pratica, le liste di nomi fra i quali cercare sono spesso molto più lunghe. L'elenco del telefono di Londra, per esempio, contiene all'incirca 2 milioni di nomi. Se il programma dovesse cercare nell'intero elenco da cima a fondo, occorrerebbe effettuare 2 milioni di confronti. Ciò richiederebbe un tempo lunghissimo anche alle estreme velocità dei computer.
Fortunatamente ci sono delle scorciatoie per eseguire queste elaborazioni. Si supponga che i nomi della lista siano "selezionati" o riordinati in ordine alfabetico crescente. È possibile usare questo fatto per organizzare la ricerca. Per esempio, è possibile iniziare confrontando il bersaglio con un nome prossimo al centro della lista. Si potrebbe essere abbastanza fortunati e scoprire un abbinamento; ma se ciò non succede, si verifica uno dei due risultati seguenti:

(a) La parola bersaglio è *minore* (e cioè più vicina all'inizio della lista) della parola centrale

oppure

(b) La parola bersaglio è *maggiore* (e cioè più vicina al termine della lista) della parola centrale

Nel primo caso è possibile essere sicuri che se il bersaglio si trova nella lista, si troverà nella prima metà. Analogamente il secondo caso dice che il bersaglio può essere soltanto nella seconda metà. In entrambi i casi si è fatto in modo di eliminare metà della lista con due confronti — uno di uguaglianza e uno di ordine relativo.
Una volta conosciuta quale metà della guida usare, è possibile applicare lo stesso processo a quella metà e identificare un *quarto* della lista originale e quindi un ottavo e così via.
Illustriamo ora il processo. Si supponga che la lista di nomi sia:

ANDREW
ANTONIA
BEATRICE
CHRIS
FRANCES
HENRY
JIM
JOAN
JULIA
OLIVE
PETER
SUSAN
TIMOTHY
TOM
WILLIAM

Si userà TOM come parola bersaglio. Per cominciare, lo confrontiamo con il nome intermedio della lista che è JOAN. Ora TOM<>JOAN, cosicchè non registriamo alcun abbinamento. Inoltre, TOM>JOAN, così possiamo eliminare tutta la lista da JOAN in su e concentrare la ricerca nella parte da JULIA fino alla fine.
La parola centrale in questa parte è SUSAN. TOM>SUSAN, cosicchè scartiamo di nuovo tutta la lista salvo il pezzettino tra TIMOTHY e WILLIAM.

La parola centrale della rimanente sezione è TOM, che dà un abbinamento diretto. Se la parola bersaglio non compare nella lista, tutto diventa rapidamente ovvio in quanto la dimensione della lista da cercare si riduce a nulla. Per esempio, se si prende il bersaglio GEORGE:

fase 1: GEORGE<JOAN, cosicchè usiamo la lista ANDREW–JIM (7 nomi)

fase 2: GEORGE>CHRIS, cosicchè usiamo la lista FRANCES–JIM (3 nomi)

fase 3: GEORGE<HENRY, cosicchè usiamo lista FRANCES–FRANCES (1 nome)

fase 4: GEORGE>FRANCES. Non è possibile alcuna ulteriore suddivisione per cui GEORGE non può essere nella lista.

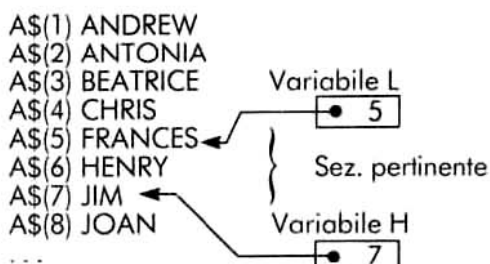
A questo punto scegliere alcuni nomi, alcuni nella lista altri no e ripetere il processo di ricerca seguendo il metodo appena descritto.

LA RICERCA DICOTOMICA

Se ci si pensa, si vedrà che ad ogni fase, la dimensione della lista da cercare viene all'incirca dimezzata. Ne segue che se si raddoppia la dimensione della lista, si aggiunge soltanto uno stadio al processo di ricerca. Quando ci si muove in liste più grandi, si inizia ad acquistare un enorme vantaggio rispetto ai metodi che si basano sulla ricerca dall'alto al basso, osservando ogni nome. Il metodo "veloce" richiede all'incirca 12 confronti per una lista con 1000 nomi o 21 per una lista con 1 milione. Dato che esso si basa sulla divisione della distinta, il metodo è detto "ricerca dicotomica".

Codifichiamo ora il metodo in BASIC. Supponiamo che la lista da esaminare contenga 100 voci e si possa trovare negli elementi da A$(1) a A$(100) della matrice A$. La parola bersaglio è X$. Per organizzare il processo occorre identificare la parte della lista in cui deve essere svolta la ricerca. Per far ciò, si useranno due variabili come *puntatori*.

> H 'punta' alla parte superiore della parte pertinente (che è cioè l'elemento con l'indice più elevato)
>
> L 'punta' alla base della pertinente sezione (l'elemento con l'indice più basso)

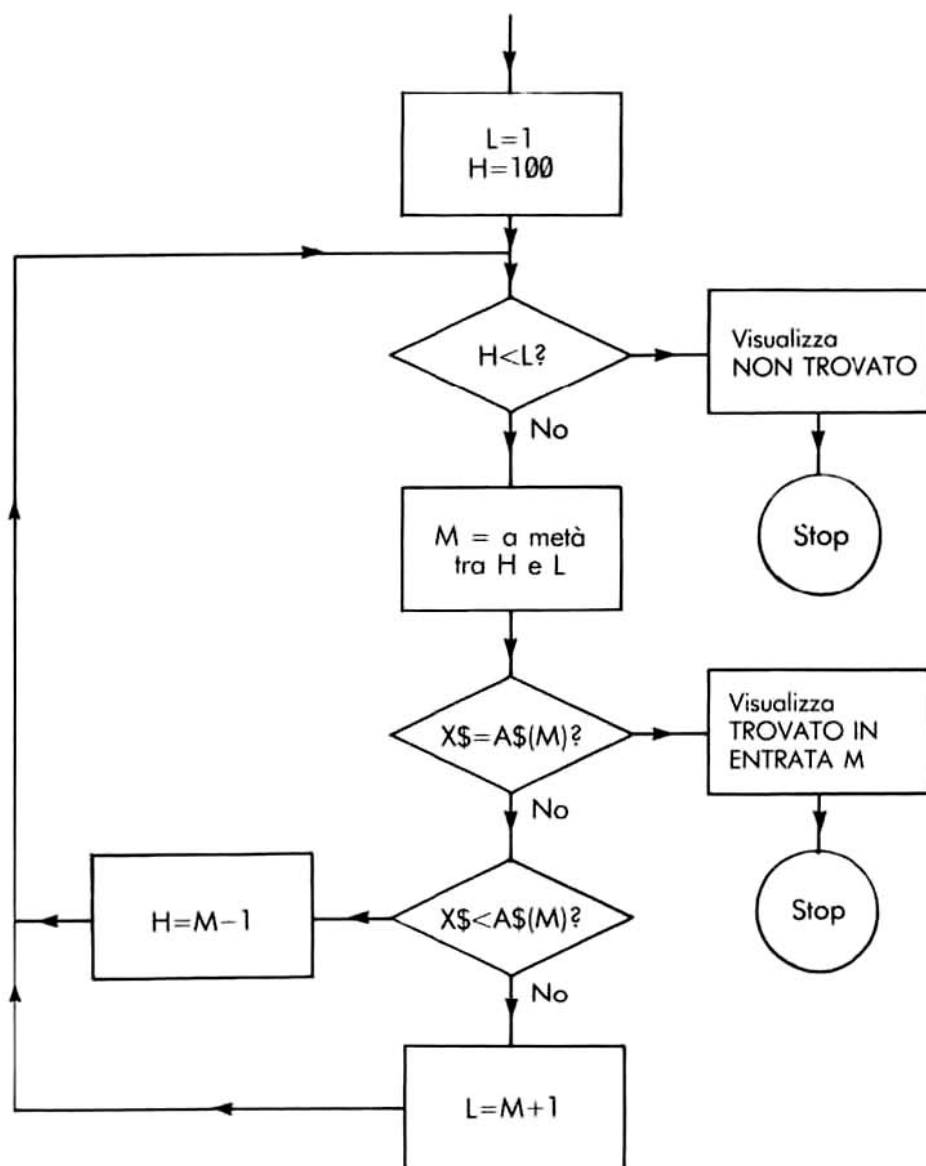
La frase "punta a" significa "contiene l'indice di". Ciò è illustrato qui di seguito:

Pertanto la parte "interessante" della lista inizia da A$(L) e termina ad A$(H). Se al limite si trova che H è minore di L, la dimensione della lista è 0 e la ricerca è fallita.

Trovare la parola centrale della parte interessante è abbastanza facile. Il suo indice è la "media" di H e L, ridotto a un numero intero se necessario. L'appropriata espressione è

```
INT(0.5*(H+L))
```

È come assegnare questo valore ad una variabile M.

Nella pianificazione dell'algoritmo dobbiamo pensare accuratamente alla modifica dei valori di H e L. Si supponga di trovare che la parola bersaglio è maggiore della parola centrale A$(M). Ciò ci fornisce un nuovo limite inferiore di L=M+1, ma non cambia per nulla il limite superiore H. Analogamente se la parola bersaglio è minore di A$(M) il nuovo limite superiore H sarà M−1, ma L non dovrà essere cambiato.

È possibile riprodurre queste idee in uno schema di flusso:

Glossario

X$: Parola bersaglio
A$(1-100): Lista di parole in cui cercare (in ordine alfabetico)

L,H: Puntatori alla parte attiva della lista A$
M: Punto intermedio della parte attiva della lista

E un corrispondente frammento di codice potrebbe essere:

```
230 L=1: H=100
240 IF H<L THEN PRINT X$; "NON
    TROVATO":STOP
250 M=INT(0.5*(H+L))
260 IF X$=A$(M) THEN PRINT X$;
    "TROVATO IN ENTRATA";M:STOP
270 IF X$<A$(M) THEN H=M-1:GOTO
    240
280 L=M+1:GOTO 240
```

ESPERIMENTO 22·1

Trasformare il codice di ricerca in una subroutine con le seguenti specifiche:

Specifiche della subroutine

Scopo: Cercare nella lista ordinata A$ tra le voci H1 e L1, l'entrata X$.

Righe: 6000-6100
Parametri di input: H1: Limite superiore di ricerca
L1: Limite inferiore di ricerca
X$: Parola bersaglio
Output: Se si trova una copia di X$ in A$, M1 è il suo indice. Se non si trova una copia M1=1

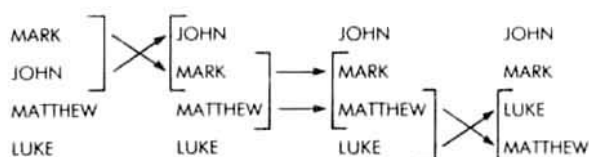
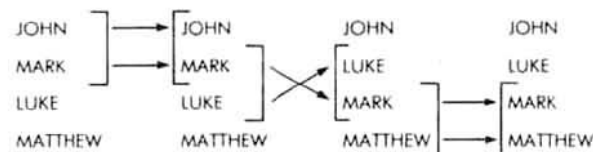
Provare la subroutine con il seguente programma di gestione:

```
10 DATA BAIN, BEAVIS, BOWEY, BURNS,
   CLARK, FLEMING
20 DATA GORDON, GREEN, HOOD,
   KIDD, MACCABE, MALLY
30 DATA MARSHALL, MILLER, NORTH,
   PACK, PERKINS, REED, ROSE
40 DATA ROSS, SIMPSON, SMITH,
   SYKES, TEDFORD, WEBSTER, WOOD
50 DIM A$(26)
60 FOR J=1 TO 26:READ A$(J):NEXT J
70 INPUT"BATTI UN NOME"; X$
80 L1=1:H1=26:GOSUB 6000
90 IF M1=-1 THEN PRINT X$; "NON
   TROVATO":GOTO 70
100 PRINT X$; "TROVATO ALLA
   POSIZIONE"; M1
110 GOTO 70
```

Esperimento 22.1 completato

METODO DI ORDINAMENTO "BUBBLE SORT"

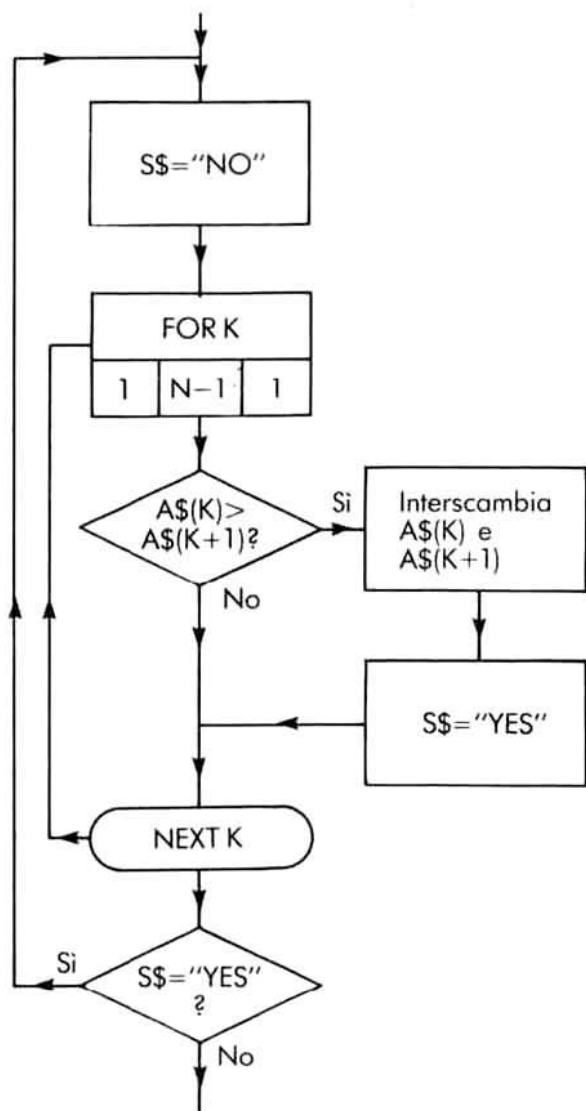
Finora in tutti gli esempi i nomi erano comodamente in ordine alfabetico quando il programma iniziava. Si supponga ora che i nomi vengano forniti in ordine casuale. Occorre pertanto ridisporli o riordinarli.
In un elenco riordinato di nomi, è possibile prendere qualsiasi coppia: quella con l'indice maggiore sarà alfabeticamente maggiore o uguale a quella con l'indice minore. Questo fatto è la base del metodo di ordinamento "Bubble sort" che scambia una coppia di numeri fuori sequenza sostituendo l'uno con l'altro.

Si inizia con una lista di nomi in ordine casuale, la si esamina e si confrontano le successive coppie di nomi (1 e 2, 2 e 3 e così via). Se si trova qualsiasi coppia fuori sequenza, questi nomi vengono scambiati; ciascuno viene spostato nello spazio precedentemente occupato dall'altro.
Ecco un esempio di tale spostamento:

Questa operazione porterà sempre il nome più grande in basso ma non lascerà necessariamente l'intera lista in ordine. Occorre riesaminarla di nuovo ripetutamente fino a che non si devono più effettuare altri spostamenti. In questo caso il secondo esame darebbe:

E il terzo esame non darebbe alcun interscambio indicando così che la lista è in ordine.
L'interscambio di due variabili non è altrettanto semplice quanto sembra. Se si cerca di scambiare i valori di X e Y scrivendo

X=Y : Y=X

la cosa non funziona; il primo comando distrugge il valore iniziale di X ed entrambe le variabili terminano con il valore originale di Y. Occorre una terza variabile temporanea — ad esempio D — per contenere il valore di X fino a che occorre:

D=X : X=Y : Y=D

E lo schema di flusso per il metodo di ordinamento "Bubble sort" è il seguente:

Glossario

S$: Segnalatore per interscambi
A$(N-1): Matrice di parole da riordinare
N: Numero di parole da riordinare
K: Puntatore a A$

La codifica corrispondente:

```
....
130 S$="NO"
140 FOR K=1 TO N−1
150 IF A$(K)>A$(K+1) THEN D$=A$(K):
    A$(K)=A$(K+1):A$(K+1)=D$:
    S$="YES"
160 NEXT K
170 IF S$ = "YES" THEN 130
....
```

ESPERIMENTO 22·2

Trasformare il metodo di ordinamento "Bubble sort" in una subroutine con la seguente specifica:

Specifiche di subroutine

Scopo: Riordinare campi in ordine alfabetico

Numeri di riga: 6500-6580

Parametri: *Input:* Lista di voci da riordinare in A1$(1) fino a A1$(N1)
Output: La lista riordinata compare da A1$(1) ad A1$(N1)

Variabili locali: KK, SS$, DD$

Provare con propri dati.

Esperimento 22.2 completato	

QUICKSORT

Il metodo di ordinamento "Bubble sort" è interessante e semplice se la lista è abbastanza corta (ad esempio 10 voci o meno) ma come la lista cresce, ogni esame diventa sempre più lungo e occorrono sempre ulteriori passate, cosicchè il tempo richiesto per il lavoro cresce col quadrato del numero di voci da riordinare. Ciò significa che una lista di 50 voci richiederà circa 25 volte il tempo necessario per selezionare una di 10.

Esistono parecchi metodi di selezione o di riordino che sono molto più veloci di questo. Uno di essi è chiamato "Quicksort" è stato inventato da C.A.R. Hoare. In termini grossolani, il tempo che esso richiede cresce soltanto con il numero di elementi da riordinare. Quicksort usa una tecnica di programmazione detta *recorsione* in cui una subroutine richiama se stessa per svolgere una parte del proprio lavoro. Molti trovano il metodo duro da comprendere particolarmente se è espresso in BASIC che non è stato progettato pensando a programmi recorsivi. Per usare efficacemente Quicksort non occorre comprenderlo; ciononostante qui c'è una breve spiegazione che si riferisce alla codifica che viene data.

Il metodo inizia con una lista di voci che non sia ordinata in alcun modo speciale. Esso prende quello inferiore, lo chiama elemento "chiave" e lo dispone nel suo posto finale corretto nella lista, assicurandosi che tutti gli elementi al di sopra di esso gli siano inferiori e tutti gli elementi al disotto gli siano superiori. Ciò avviene interscambiando le voci o gli elementi se necessario. Per esempio, qui di seguito è indicata la prima fase nel riordino di una lista di 8 elementi:

5	5	
18	4	Tutti elementi infer. a 12
23	6	
4	→12	Posiz. corretta per 12 (chiave)
6	18	
17	23	
37	17	Tutti elementi super. a 12
(12)	37	

La seconda fase consiste nel riordinare tutti gli elementi al disopra dell'elemento chiave e la terza fase selezionare tutti quelli aldisotto dell'elemento chiave. Per entrambe queste fasi, la subroutine richiama se stessa recorsivamente, per cui il riordino di una parte di una lista non diventa altro che il problema di riordinarla tutta. Una buona parte della subroutine che segue è provvista dal meccanismo della recorsività. La matrice SS% e la variabile puntatore PP sono usate per ricordare esattamente ciò che sta accadendo a qualsiasi "livello" di controllo cosicchè tutte le chiamate e i ritorni avvengano in maniera ordinata. Il comando 6170 è l'equivalente di un RETURN.

Nella subroutine, le righe da 6040 a 6090 eseguono la fase 1; quelle da 6100 a 6130 si occupano della fase 2 (che può essere saltata se la "lista" al disopra dell'elemento chiave comprende meno di 2 elementi). Le righe da 6140 a 6160 agiscono alla fine della fase 3 e le righe 6010, 6020, 6110, 6130, 6150 e 6170 sono tutto ciò che occorre per la recorsione. La subroutine comprende due aspetti non familiari: un nome di matrice che termina con % e un comando con la parola chiave ON. Entrambi verranno discussi più avanti.

Specifiche della subroutine

Scopo: Riordinare gli elementi in ordine numerico, usando l'algoritmo Quicksort di Hoare

Numeri di riga: da 6000 a 6180

Parametri: *Input:* Lista di numeri da riordinare in A1(1) fino a A1(N1). Numero degli elementi in N1
Output: La lista riordinata compare da A1(1) a A1(N)

Variabili locali: SS, SS%, AA, BB, XX, YY, ZZ, DD, PP

NOTE: (i) SS% non deve essere usato altrove nel programma se la subroutine è richiamata più di una volta.
(ii) La subroutine può essere usata per riordinare stringhe invece di numeri se vengono effettuate al suo interno le seguenti sostituzioni:

A1$ per A1; ZZ$ per ZZ; DD$ per DD

```
6000 REM QUICKSORT:RIORDINA N1
     ELEMENTI DI A1
6010 IF SS=1 THEN 6030
6020 DIM SS%(N1):SS=1:REM DICHIARA
     CATASTA
6030 AA=1:BB=N1:SS%(0)=1:PP=1
6040 XX=AA: YY=BB: ZZ=A1(BB)
6050 IF XX>=YY THEN 6090
6060 IF A1(XX)<=ZZ THEN XX=XX+1:
     GOTO 6050
6070 IF A1(YY) >=ZZ THEN YY=YY-1:
     GOTO 6050
6080 DD=A1(YY):A1 (YY) = A1 (XX):
     A1(XX)=DD: GOTO 6050
6090 A1(BB) = A1(XX):A1(XX)=ZZ
6100 IF XX-AA<= 1 THEN 6140
6110 SS%(PP)= XX: SS%(PP+1)=BB:
     SS%(PP+2)=2: PP=PP+3
6120 BB=XX-1: GOTO 6040
6130 PP=PP-3: XX=SS%(PP):
     BB=SS%(PP+1)
```

```
6140 IF BB−XX<=1 THEN 6170
6150 SS%(PP)=3: PP=PP+1: AA=XX+1:
     GOTO 6040
6160 PP=PP−1
6170 ON SS%(PP−1) GOTO 6180, 6130,
     6160
6180 RETURN
```

Questa completa subroutine intitolata "QUICKSORT" può essere trovata sulla cassetta di nastro.

CONFRONTO DEI TEMPI DI RIORDINO

Quicksort è così più complicato di Bubble sort che ci si potrebbe domandare se vale la pena di usarlo. È possibile giudicare da soli da questa tabella che mostra i tempi necessari per riordinare matrici di varie dimensioni. Le cifre sono state trovate eseguendo entrambi i tipi di riordino su un **64** e cronometrandoli.

Dimens. matrice	*Tempo (Quicksort)*	*Tempo (Bubble sort)*
20	2	5
40	5	22
60	8	47
80	14	93
100	17	138
120	20	192
140	24	282
160	27	357
180	31	445
200	37	569

LA MEMORIA DEL COMMODORE 64

Confrontato con la maggior parte degli altri personal computer, il Commodore **64** ha una capacità di memoria notevole. Ciò nonostante, quando si comincia ad usare le matrici, ci si può al limite trovare a corto di spazio anche col COMMODORE **64**. Ciò in quanto le matrici "arraffano" molto rapidamente una grande quantità di spazio; ciascun elemento di una matrice numerica usa fino a 5 byte di memoria e ogni elemento stringa usa 3 byte, più lo spazio necessario per la stringa stessa. Ci sono poi piccoli sovraccarichi addizionali per ciascuna matrice.
In questa sezione osserveremo il modo in cui la memoria del **64** è organizzata. Un indicatore utile è la funzione incorporata FRE(Ø) che dice in qualsiasi momento quanti byte rimangono inutilizzati. Quando si accende il **64** per la prima volta, compare il messaggio:

64K RAM SYSTEM 38911 BASIC BYTES FREE

(Come si sa, il Commodore **64** ha un totale di 65536 byte di RAM ma un buon numero di essi sono allocati per altri scopi).
Ora se si batte

PRINT FRE(Ø)

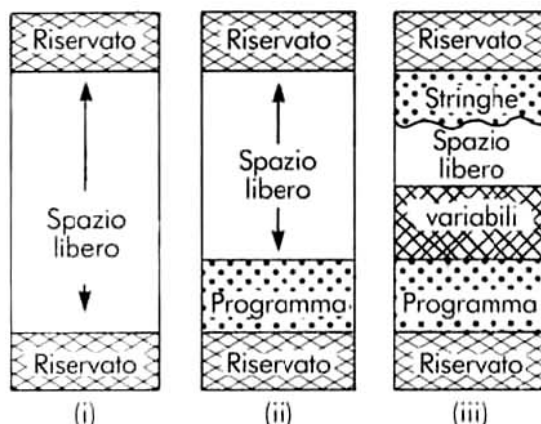
La macchina risponde -26627.

Quando viene usata la funzione FRE(Ø), sul **64** si ha frequentemente un risultato negativo a causa del modo in cui il sistema operativo calcola il risultato della funzione. Quando si chiede FRE(Ø) viene calcolato un risultato intero. Gli interi possono contenere qualsiasi valore da -32768 a +32767. Per comprendere a fondo perché ciò fa sì che la funzione FRE abbia risultati negativi occorrerebbe una lunga discussione teorica. Basterà dire che un numero molto grande (maggiore di 32767) diventa negativo, per cui per calcolare il numero esatto di "byte liberi" occorre aggiungere 65536 al risultato.

es. Se FRE(Ø) dà -26627, aggiungendo
65536 si ottiene 38909

Si tratta di un valore minore di quello che era presente all'accensione in quanto la funzione FRE(Ø) usa due byte per eseguire i suoi calcoli.
Una formula generale per calcolare i "byte liberi" effettivi è:

liberi=(−65536★(FRE(Ø)<Ø)+FRE(Ø)+2

La situazione generale è illustrata nella parte (i) dello schema che segue. Dei 65536 byte nel Commodore **64**, 26625 sono riservati per vari scopi, 2 byte sono usati per eseguire la funzione FRE ma i restanti sono ancora liberi.

Successivamente si potrebbe battere un programma o caricarne uno da una cassetta. Il programma è inserito nella parte inferiore della sezione libera della memoria, occupando all'incirca 1 byte per ogni carattere. Il risultato è indicato nel diagramma parte (ii).
Ora si inizia il programma. La macchina inizia a rispondere ai comandi e non appena incontra qualsiasi variabile cui si fa riferimento per la prima volta, alloca lo spazio necessario nell'area immediatamente adiacente al programma stesso. Ai comandi DIM è attribuito uno spazio nella stessa area e lo spazio può esaurirsi rapidamente; un comando apparentemente innocente tipo

DIM A(200)

costerà oltre 1000 byte.

Una volta allocato lo spazio per una variabile o una matrice, questo non può essere recuperato e usato per qualsiasi altro scopo fino a che il programma non è interrotto. Le stringhe vengono gestite in maniera diversa. Le stringhe che vengono lette direttamente dalle istruzioni DATA nel programma non occupano alcun spazio in più. Le stringhe che vengono lette dalla tastiera o costruite con "+", MID$ e altre funzioni stringa sono disposte all'altra estremità della memoria, lasciando spazio libero tra le stringhe e le variabili. Ciò è illustrato nella parte (iii) del diagramma. Lo spazio usato dalle stringhe è recuperabile; quando una stringa non è più necessaria, può essere scartata e lo spazio è restituito all'area libera.
(Se si pensa che questo sia un processo complicato, si è nel giusto — esso è detto "raccolta dei rifiuti". Fortunatamente è completamente automatico e non occorre saperne nulla). A parte il limite della dimensione, la memoria del **64** non è partizionata in alcun modo. È possibile avere programmi, variabili e stringhe a piacere posto che lo spazio totale non superi quello libero disponibile.
Impostare ed eseguire i seguenti programmi e pensare ai risultati alla luce di quanto detto:

```
 10 PRINT "SPAZIO LIB",
    "DOPO RIGA"
 20 PRINT FRE(0),10
 30 X=0
 40 PRINT FRE(0), 30
 50 DIM A(20)
 60 PRINT FRE(0),50
 70 DIM N$(5)
 80 PRINT FRE(0),70
 90 C$="UNA STRINGA"
100 PRINT FRE(0),90
110 D$="ALTRA" + "STRINGA"
120 PRINT FRE(0),110
130 D$=""
140 PRINT FRE(0),130
150 C$=""
160 PRINT FRE(0),150
170 STOP
```

MATRICI BIDIMENSIONALI

Come è possibile vedere, le matrici sono utili nei problemi in cui il programma deve gestire contemporaneamente molte variabili diverse. In qualche problema è naturale disporre queste variabili in una tabella quadrata o rettangolare anziché in una semplice lista ordinata. Si consideri un programma per giocare a scacchi. Esso deve "conoscere" quale pezzo occupa eventualmente ciascuna casella della tastiera. Ogni quadrato o casella può essere ripetuto da una variabile il cui valore riflette il pezzo di quella casella. Le 64 variabili sono disposte in una tabella con 8 file e 8 colonne, che formano il profilo della scacchiera. Il BASIC consente l'uso di matrici bidimensionali (e anche tridimensionali o più). Una tipica dichiarazione di una matrice bidimensionale sarebbe la seguente:

```
DIM X(5,7)
```

Questo comando imposta una matrice denominata X in cui ogni elemento è un numero. La matrice ha (5+1) ossia 6 file e (7+1) ossia 8 colonne: 48 elementi in tutto. Ecco un'immagine di tale matrice:

	0	1	2	3	4	5	6	7
0	X(0,0)	X(0,1)	X(0,2)	X(0,3)	X(0,4)	X(0,5)	X(0,6)	X(0,7)
1	X(1,0)	X(1,1)	X(1,2)	X(1,3)	X(1,4)	X(1,5)	X(1,6)	X(1,7)
2	X(2,0)	X(2,1)	X(2,2)	X(2,3)	X(2,4)	X(2,5)	X(2,6)	X(2,7)
3	X(3,0)	X(3,1)	X(3,2)	X(3,3)	X(3,4)	X(3,5)	X(3,6)	X(3,7)
4	X(4,0)	X(4,1)	X(4,2)	X(4,3)	X(4,4)	X(4,5)	X(4,6)	X(4,7)
5	X(5,0)	X(5,1)	X(5,2)	X(5,3)	X(5,4)	X(5,5)	X(5,6)	X(5,7)

Ciascun elemento della matrice ha due indici: un numero di fila e uno di colonna. Per esempio X(3,4) è nella fila 3 e nella colonna 4. A parte questa fondamentale differenza, tutto ciò che si conosce a proposito delle matrici unidimensionali, può essere esteso a quelle bidimensionali. Passiamo ad un'*illustrazione*. Si supponga di aver eseguito un'indagine e scoperto il prezzo di alcune merci base in ciascuno dei cinque negozi della propria zona. Si potrebbe esprimere i risultati in una tabella di questo genere

	FINE FARE	ASDA	SAINS-BURYS	CO-OP	FRASERS
FARINA	29	31	27	26	32
PATATE	15	12	13	24	33
BURRO	47	49	40	45	39
ZUCCHERO	22	20	19	27	29
FORMAGGIO	94	80	103	107	99
MELE	32	18	22	27	21
	(Tutti i prezzi sono espressi in Lire all'ettogrammo)				

Il problema da risolvere è, data una particolare lista di acquisti, quale è il negozio più a buon mercato da visitare? Un'immagine "dell'utente" del programma potrebbe essere:

```
INDICA FABBISOGNO
IN ETTI.

FARINA?       3
PATATE?      14
BURRO?        1
ZUCCHERO?     0
FORMAGGIO?    2
MELE?         3
```

```
MEGLIO COMPRARE DA ASDA
DOVE PAGHI
3 ETTI FARINA      :  93
14 ETTI PATATE     : 168
1 ETTO BURRO       :  49
0 ETTI ZUCCHERO    :   0
2 ETTI FORMAGGIO: 160
3 ETTI MELE        :  54
      TOTALE       : 524
```

L'algoritmo base è lineare:

Iniziamo a scegliere alcune variabili e i loro nomi. Certamente avremo necessità di memorizzare il nome dei negozi e i vari articoli delle merci. Le variabili adatte sono:

da F$(1) a F$(6) per i cibi
e da S$(1) a S$(5) per i negozi

Ora, abbiamo bisogno di matrici per mostrare la quantità di ciascun cibo necessario e il corrispondente importo pagato (o totale) presso ciascun negozio. Le variabili adatte sono:

da F(1) a F(6) per le quantità e da T(1) a T(5) per i totali.

Infine, useremo una matrice bidimensionale per contenere la tavola dei prezzi. Una dichiarazione del tipo

DIM P(6,5) potrebbe andare bene.

Notare che abbiamo sistematicamente ignorato elementi con indici di 0. Ciò è comune nei piccoli problemi.
Il codice effettivo è abbastanza semplice, anche se un pochino lungo. Eccolo:

```
 10 DATA FARINA, PATATE, BURRO,
    ZUCCHERO, FORMAGGIO, MELE
 20 DATA FINE FARE, ASDA,
    SAINSBURYS, COOP, FRASERS
 30 DATA 29, 31, 27, 26, 32
 40 DATA 15, 12, 13, 24, 33
 50 DATA 47, 49, 40, 45, 39
 60 DATA 22, 20, 19, 27, 29
 70 DATA 94, 80, 103, 107, 99
 80 DATA 32, 18, 22, 27, 21
 90 DIM F$(6),S$(5), F(6), T(5), P(6,5)
100 FOR K=1 TO 6: READ F$(K): NEXT K
110 FOR J=1 TO 5: READ S$(J): NEXT J
120 FOR K= 1 TO 6: FOR J=1 TO 5
130 READ P(K,J)
140 NEXT J,K
150 PRINT "SHIFT e CLR HOME PREGO
    INDICARE FABBISOGNO"
160 PRINT "IN ETTI"
170 FOR K=1 TO 6
180 PRINT F$(K);: INPUT F(K)
190 NEXT K
200 FOR J=TO 5
210 FOR K=1 TO 6
220 T(J)=T(J)+F(K)★P(K,J)
230 NEXT K,J
240 M=T(1): N=1
250 FOR J=2 TO 5
260 IF T(J)<M THEN M=T(J): N=J
270 NEXT J
280 PRINT "MEGLIO COMPRARE"; S$(N)
290 PRINT "DOVE PAGHI": PRINT
300 FOR K=1 TO 6
310 PRINT F(K);"ETTI";
320 IF F(K)<>1 THEN PRINT "SHIFT
    e CRSR SHIFT e CRSR S.";
325 X=F(K)★P(K,N)
330 PRINT F$(K);TAB(15)
```

```
340 PRINT LEFT$(RIGHT$(" SPACE
     11 volte"+STR$(X),10),10)
345 NEXT K
350 PRINT:PRINT"TOTALE=";TAB(15);:

     PRINT LEFT$(RIGHT $(" SPACE
     11 volte" + STR $(M),10),10)
360 STOP
```

Devono essere chiariti uno o due punti di scarsa importanza.

(a) Tutte le matrici sono dichiarate insieme in un comando. Ciò è più breve che scrivere

```
 90 DIM F$(6)
100 DIM S$(5)
```

e così via.
Il limite al numero di matrici che possono essere dichiarate è definito dalla lunghezza massima di riga: 80 caratteri.

(b) La sequenza dei comandi

```
NEXT J
NEXT K
```

può essere compressa in

```
NEXT J,K
```

Ciò si applica altrettanto bene a qualsiasi variabile di controllo e a qualsiasi numero di esse (quantunque sia raro il caso di trovarne più di due).

(c) La frase "TAB (15)" nel comando 350 fa sì che la macchina muova il suo cursore interno alla 15esima colonna nello schermo (se il cursore non vi si trova già). È usato per allineare l'importo pagato per ciascun articolo alimentare.
In generale, le parentesi possono contenere qualsiasi espressione. Pertanto il programma

```
10 FOR J=20 TO 1 STEP -1
20 PRINT TAB(J);"/"
30 NEXT J
40 STOP
```

visualizzerà una linea diagonale attraverso lo schermo.

Notare che le righe 340 e 350 danno la corretta giustificazione per i valori di prezzo.
Notare che J è sistematicamente usato per il numero di negozio e K per il numero di tipo di cibo. P(K,J) è pertanto il prezzo del cibo numero K presso il negozio numero J.

Questo esperimento è in due parti:

(a) Una classe con molti studenti dà un esame competitivo. L'insegnante produce una serie di punteggi del tipo

```
ADAMS     27
BRIGGS    66
CHILVERS  89
DALE      38
.....
```

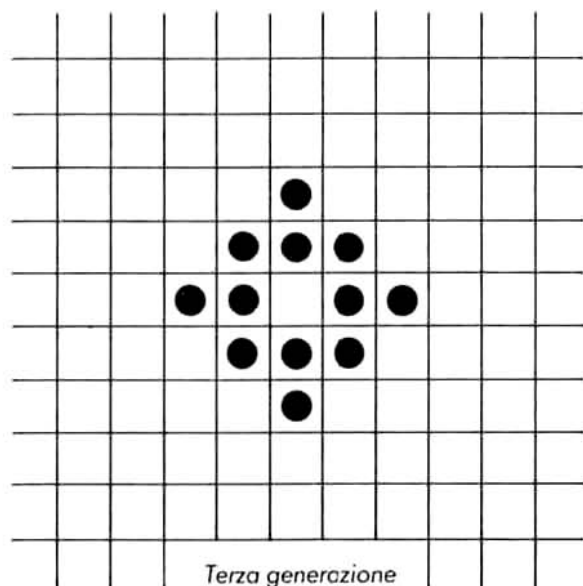
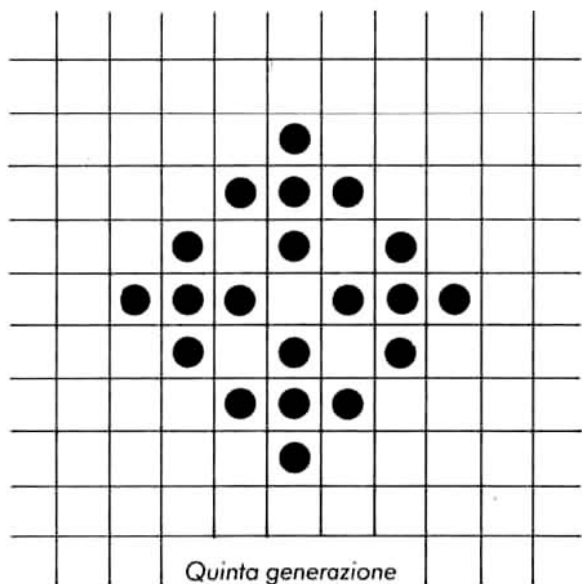
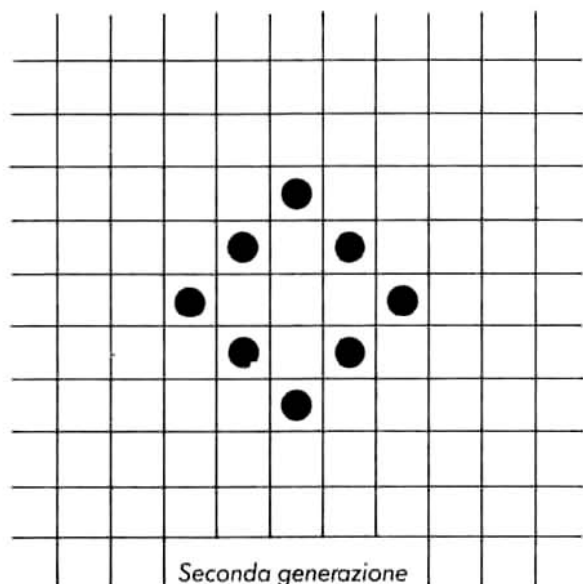
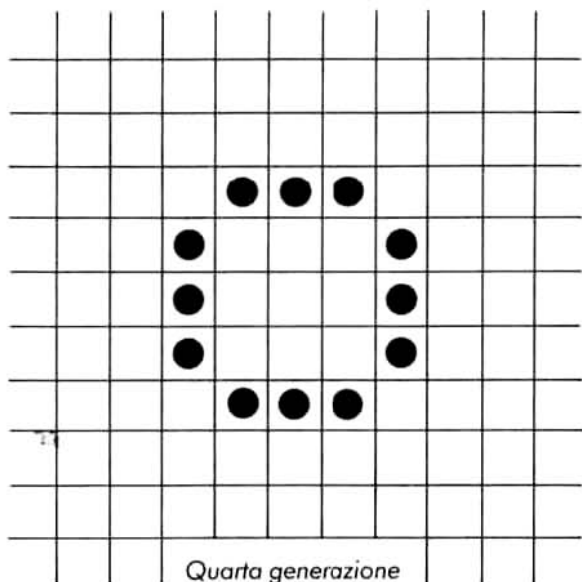
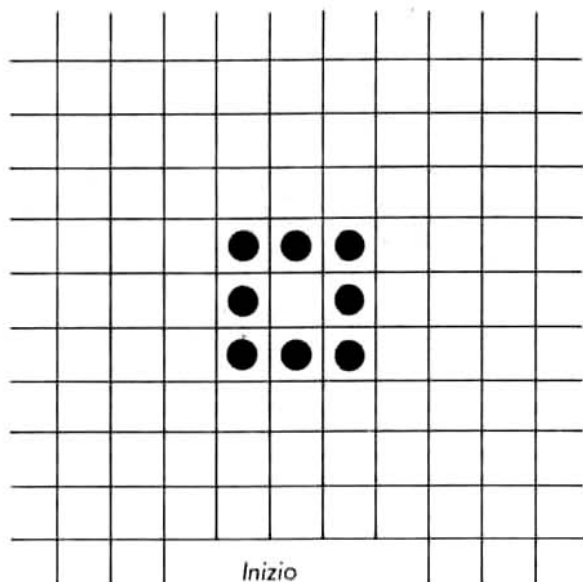
e così via.
Le regole dicono soltanto il primo 25% (un quarto) degli studenti può passare l'esame.
Scrivere un programma che può leggere nella lista dei punteggi originali e visualizzare i nomi degli studenti che passano. Si supponga che non ci siano più di 100 studenti e che il voto dell'ultimo studente sia seguito dal nome fittizio "ZZZZ".
SUGGERIMENTO: riordinare una copia dei punteggi usando la subroutine QUICKSORT sulla cassetta di nastro e trovare il punteggio minimo abilitante a un quarto di strada lungo l'elenco riordinato. Usarlo per individuare gli studenti che superano l'esame.

(b) Il gioco della "vita" è stato inventato da R. Convay, un matematico inglese. Esso riguarda la storia della vita di una colonia di piccoli insetti che vivono in un'area rettangolare, uno per ogni cella.
La colonia vive da generazione in generazione. Il fato di ciascun insetto è determinato dalle seguenti regole:

1. Se un insetto ha 1 o meno vicini immediati, muore di solitudine.

2. Se ha 4 o più vicini, muore di sovraffollamento.

3. Se ha 2 o 3 vicini, sopravvive alla successiva generazione.

Inoltre, se una cella vuota ha esattamente 3 insetti nelle celle vicine, nasce un nuovo insetto in quella cella.
Per dare un esempio, si consideri

Scrivere un programma per svolgere il gioco "della vita" su una matrice 9x9. Ciascun elemento dovrebbe essere una stringa che contiene un carattere (ad esempio "★" o uno spazio). Il programma dovrebbe leggere una "posizione di partenza" quindi visualizzare le successive generazioni fino a che non viene interrotto.
Si tratta di un esperimento sfidante che consente di usare il programma parziale "LIFERSTAR" sulla cassetta di nastro. Il programma legge una posizione di partenza nella matrice X$(9,9) (righe 10-180) quindi la visualizza (righe 190-260). Il prògramma parziale dichiara anche una matrice Y$(9,9) che si troverà utile per passare da una generazione alla successiva.

SUGGERIMENTO (1):

Per determinare la successiva generazione, usare la matrice Y$, che è già stata dichiarata. Quando si è costruita una generazione completa in Y$, copiarla di nuovo in X$.

SUGGERIMENTO (2):

Se si osserva la cella X$ (J,K) (dove J e K sono indici), le otto celle vicine saranno:

X$(J−1,K−1), X$(J−1,K), X$(J−1,K+1)

X$(J,K−1) X$(J,K+1)

X$(J+1,K−1), X$(J+1,K), X$(J+1,K+1)

Per impedire riferimenti a celle che non sono per nulla nella matrice, si assuma che le celle del margine siano sempre vuote e si limitino le operazioni alle 7 file e colonne "interne".

Controllare ora la risposta a fronte di quella indicata nell'Appendice C.

Esperimento 22.3 completato	

UNITA':23

Uno sguardo da vicino all'interno del COMMODORE 64	PAGINA 233
Bit, Byte e indirizzi	233
La struttura del COMMODORE 64	234
La memoria del COMMODORE 64	235
Il comando PEEK	236
Esperimento 23.1	237
Il comando POKE	238
Esperimento 23.2	241
I caratteri in SET 2	242
Definizione di propri caratteri	242
Altro a proposito di PEEK e POKE	244
Un esempio di animazione	244
Esperimento 23.3	251

UNO SGUARDO DA VICINO ALL'INTERNO DEL COMMODORE 64

Un computer è un dispositivo estremamente complicato. Se si cerca di spiegare tutto quanto lo riguarda ad un principiante, in una sola volta, lo si lascia frastornato e inevitabilmente confuso prima che egli possa fare qualcosa d'interessante o di utile. Per contro, è possibile trattare la macchina come un pacchetto ad un party di ragazzi: qualcosa con numerosi involucri di carta che si possono strappare uno alla volta. Se si nascondono i dettagli non necessari, è sempre possibile far sì che lo strato più esterno appaia abbastanza semplice. Per esempio, molte persone penseranno sempre al COMMODORE **64** esattamente come ad una macchina che esegue giochi che vengono forniti su una cassetta di nastro o su cartucce a innesto. Per coloro che non conoscono nulla di programmazione ciò è perfettamente ragionevole e un utile livello di conoscenza.

Alcune persone desiderano andare più a fondo. Il lettore avrà già capito che il **64** è una macchina che memorizza ed esegue programmi BASIC. Anche questo è un importante livello di conoscenza in quanto consente di usare la macchina in numerosi modi originali e interessanti, ma esclude dettagli sul modo in cui le informazioni sono memorizzate, come esegue un programma e sul modo in cui effettivamente funziona.

In questa Unità si andrà un po' più in profondità verso il meccanismo interno del **64**. Si troverà che la descrizione della memoria del **64** sembra diversa dall'immagine presentata nelle unità precedenti. Ciò in quanto si sta osservando la memoria da un punto di vista nuovo e più ravvicinato. Entrambe le descrizioni sono vere e ciascuna è appropriata al livello al quale il sistema viene descritto.

Questa unità esplora i misteriosi comandi PEEK e POKE. Occorre iniziare con due avvertenze:

1. A differenza del resto del manuale, il materiale in questa unità si applica soltanto al **64** e non può essere usato con qualsiasi altro computer. La maggior parte dei personal computer ammettono i comandi PEEK e POKE che però eseguono cose diverse, trattandosi di macchine diverse.

2. PEEK e POKE sono comandi che consentono di introdursi ulteriormente nel funzionamento interno del **64** più di qualsiasi altro comando BASIC. Ciò significa che viene escluso un livello di protezione. Un programma con errori può modificare il software del computer e fare in modo che si comporti in maniera molto strana. Ad esempio, la tastiera potrebbe risultare totalmente inattiva o sullo schermo potrebbero essere visualizzati sciami di caratteri strani. Inoltre il computer potrebbe rifiutarsi di obbedire a semplici comandi tipo "LIST" o "RUN". In questi casi, è sempre possibile riguadagnare il controllo spegnendo la macchina per 30 secondi. Dato che ciò cancella il programma, è doppiamente importante memorizzare frequentemente il programma su una cassetta se si usano i comandi PEEK e POKE.

La corruzione del software è un effetto provvisorio. È assolutamente impossibile provocare al **64** qualsiasi danno permanente eseguendo qualsiasi programma, per quanto possa essere pieno di errori.

Per conoscere il PEEK e il POKE, occorre per prima cosa imparare un po' di più sul computer **64** vero e proprio.

Ciò che il libro dice è abbastanza complicato così non occorre sorprendersi se occorre rileggere parecchie volte prima di comprendere a fondo tutti i punti più sottili.

BIT, BYTE E INDIRIZZI

L'unità fondamentale d'informazione all'interno di un computer è la cifra binaria o bit. Un bit può avere solo due possibili valori: 0 e 1.

I bit di per sè non sono particolarmente utili. Nel COMMODORE **64** (e di conseguenza nella maggior parte degli altri computer) essi sono raccolti in gruppi di 8 detti byte. In un byte ci sono 256 possibilità combinazioni di bit. Alcune di queste sono:

```
00000000   01011100   10101011   11111111
```

Il significato di un byte dipende interamente dal contesto in cui è usato. Nel COMMODORE **64** potrebbe essere:

(a) Un codice per un carattere specifico (ad esempio "A" potrebbe essere

(b) Un profilo di un massimo di 8 punti sullo schermo.

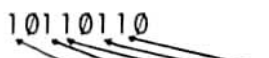
(c) Un'istruzione al microprocessore perché esegua qualche azione.

(d) Un numero, dove i bit sono interpretati da una regola matematica detta "sistema binario".

(e) Uno di un gruppo di 5 byte che insieme costituiscono il valore di una variabile numerica.

(f) Uno dei parecchi byte che rappresentano i caratteri in una variabile stringa.

Per valutare i byte è di aiuto conoscere il sistema binario. Fortunatamente non è molto difficile. Un byte da 8 bit può essere convertito in un numero normale con la seguente regola:

Il bit più a sinistra equivale a 128 se 1, a zero se 0
Il successivo bit equivale a 64 se 1, a zero se 0
Il successivo bit equivale a 32 se 1, a zero se 0
Il successivo bit equivale a 16 se 1, a zero se 0
Il successivo bit equivale a 8 se 1, a zero se 0
Il successivo bit equivale a 4 se 1, a zero se 0
Il successivo bit equivale a 2 se 1, a zero se 0
Il bit più a destra equivale a 1 se 1, a zero se 0

Per dare un esempio prendiamo il byte

10110110

I vari bit equivalgono a 128, 32,16, 4 e 2, cosicchè il numero corrispondente è 128+32+16+4+2 ossia 182.
I valori per i bit sono facili da ricordare in quanto essi iniziano a 128 e quindi ciascuno è esattamente la metà di quello che lo precede.
Talvolta occorre convertire un numero nel suo equivalente binario. Perché la conversione possa avvenire, il numero deve essere minore di 256 (e non minore di 0). Il metodo è:

1. Se è possibile sottrarre 128, farlo e scrivere un 1. Altrimenti scrivere uno 0.

2. Se è possibile sottrarre 64 dal numero rimasto dopo la fase 1, farlo e scrivere un 1 alla destra del precedente simbolo. Altrimenti scrivere uno 0.

3-8 Procedere analogamente per 32, 16, 8, 4, 2 e 1. Si avrà una fila di 8 bit che sono l'equivalente binario del numero iniziale.

Come esempio si prenda il numero 201.

(1)	201-128=73 per cui	1
(2)	73-64=9 per cui	11
(3)	9-32 non funziona, per cui	110
(4)	9-16 non funziona, per cui	1100
(5)	9-8=1 per cui	11001
(6)	1-4 non funziona, per cui	110010
(7)	1-2 non funziona, per cui	1100100
(8)	1-1=0 per cui	11001001

Così la forma binaria di "201" è "11001001".
Talvolta il computer ha bisogno di gestire gli "indirizzi". Un indirizzo è un numero costituito da due byte affiancati. L'indirizzo con il valore più elevato possibile è costituito da sedici 1: e cioè 1111111111111111. Se si estendono le regole di conversione e le si applica, si troverà che il numero equivalente è 65535. Dato che l'indirizzo più basso possibile è zero, ne segue che ci sono 65536 possibili indirizzi diversi.

LA STRUTTURA DEL COMMODORE 64

Il COMMODORE **64** ci compone di parecchie unità fra di loro collegate. Esse sono:

(a) Una *memoria* costituita da parecchie sezioni:

- *RAM* o memoria ad accesso casuale, i cui contenuti possono cambiare da un momento all'altro. La RAM è normalmente usata per memorizzare programmi BASIC e variabili.
- *ROM* o memoria di sola lettura, che contiene informazioni incorporate durante il processo di fabbricazione. I suoi contenuti non possono mai cambiare. La ROM è usata per memorizzare programmi (come lo Screen Editor) che devono essere presenti nel computer.
- *Registri di Controllo Periferiche* (o "PCR") che fanno parte dei chips di controllo suono e visione e i chip di interfaccia usati per collegare la tastiera, l'unità cassetta, l'unità disco e la stampante.
  Tutta la memoria è organizzata in gruppi di byte. Nella maggior parte dei casi la quantità di memoria in una qualsiasi categoria è un multiplo esatto di 1024 byte. Questa quantità di memoria è spesso detta 1 Kilobyte ossia brevemente 1 "K". Pertanto la memoria contiene 65536 byte di RAM ("64K"), 20480 byte di ROM ("20K") e spazio per un massimo di 4096 byte ("4K") di registri per il controllo delle periferiche.

(b) Un *microprocessore* che estrae le istruzioni dalla memoria (ROM o RAM) e le segue. La maggior parte delle istruzioni fanno sì che i contenuti della RAM o dei registri di controllo della RAM o dei registri di controllo delle periferiche vengono in qualche modo alterati. Le singole istruzioni sono all'incirca simili ai comandi in un programma BASIC ma sono più semplici. Esse sono anche eseguite molto più velocemente - alla frequenza di circa mezzo milione ogni secondo.

(c) Una *Tastiera*, di cui si occupa il Microprocessore.

(d) L'*Unità di Controllo Video*, che produce l'immagine sullo schermo TV.

(e) L'*Unità di Controllo Suono*, che sintetizza i suoni che si odono sul televisore.

(f) Il computer può disporre anche di un'*unità a cassetta* o di un'*unità disco*. Queste unità sono controllate dal Microprocessore e dai PCR.

La Figura 23.1 è uno schema della struttura complessiva del COMMODORE **64**.

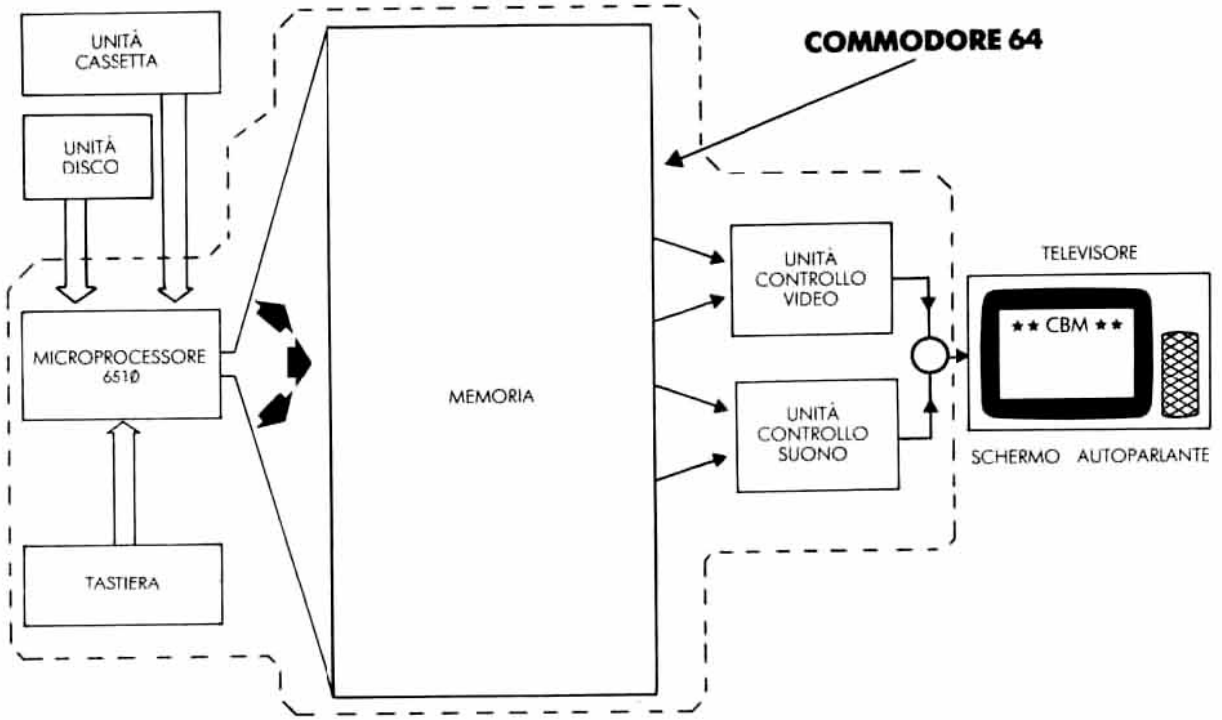



Figura 23.1

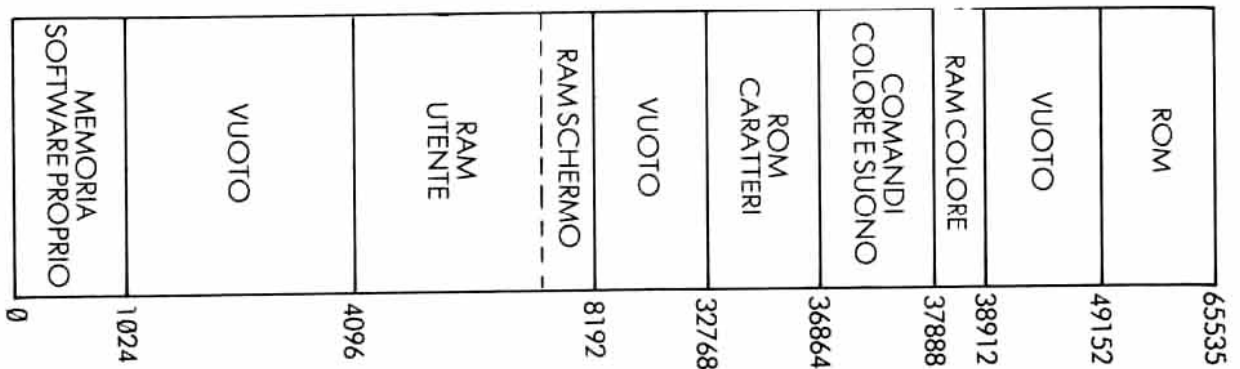



Figura 23.2

Mappa di indirizzi per COMMODORE **64** (una macchina con memoria ad uno strato)

LA MEMORIA DEL COMMODORE 64

La struttura del COMMODORE **64** è abbastanza complessa in quanto i proggettisti hanno fatto in modo di dotare la macchina di una quantità di memoria a prima vista incredibile.
Nella maggior parte dei computer ogni byte della memoria ha un proprio indirizzo esclusivo. C'è un byte con indirizzo Ø, un altro con indirizzo 1 e così via. I byte sono riuniti in gruppi con indirizzi adiacenti ed abbastanza spesso ci sono dei vuoti nella sequenza di indirizzi. Il COMMODORE VIC 20 illustra questo punto in maniera perfetta. È possibile disegnare una "mappa di indirizzi" per il VIC 20 che assomiglia a quella della Figura 23.2.
La mappa dice che c'è una RAM tra Ø e 1Ø23 e di nuovo tra 4Ø96 e 8191. La ROM, la RAM dei colori e l'unità di controllo delle periferiche riempiono le altre parti dello spazio di indirizzo. Ci sono varie zone vuote che possono essere riempite da cartucce che si inseriscono a innesto nei connettori di espansione nella parte posteriore del computer.
In tale schema, dove ciascun byte nella memoria ha un proprio indirizzo speciale, ci sono chiari vantaggi. Quando il microprocessore desidera richiamare o alterare i contenuti di una particolare cella, tutto ciò che deve fare è di indicare l'indirizzo corretto. Ciò sembrerebbe così ovvio da non valere la pena di parlarne ma è bene procedere con ordine!
Sfortunatamente, questo schema non funziona sul **64**, in quanto - come molti avranno immaginato - la quantità totale di memoria è sensibilmente più elevata del numero dei diversi indirizzi che è possibile creare con 2 byte, ossia con 16 bit.
Il modo per aggirare il problema consiste nell'usare una memoria a strati. Lo strato inferiore è costituito dai 64K di RAM, ma alcune sue parti sono nascoste da altri tipi di memoria. Lo schema che segue (Figura 23.3) mostra due viste della memoria del COMMODORE **64**. Quando il computer sta eseguendo normali programmi BASIC, esso "vede" soltanto la rappresentazione superiore, che comprende circa 40K di RAM, 20K di ROM e 4K di registri di controllo delle periferiche.

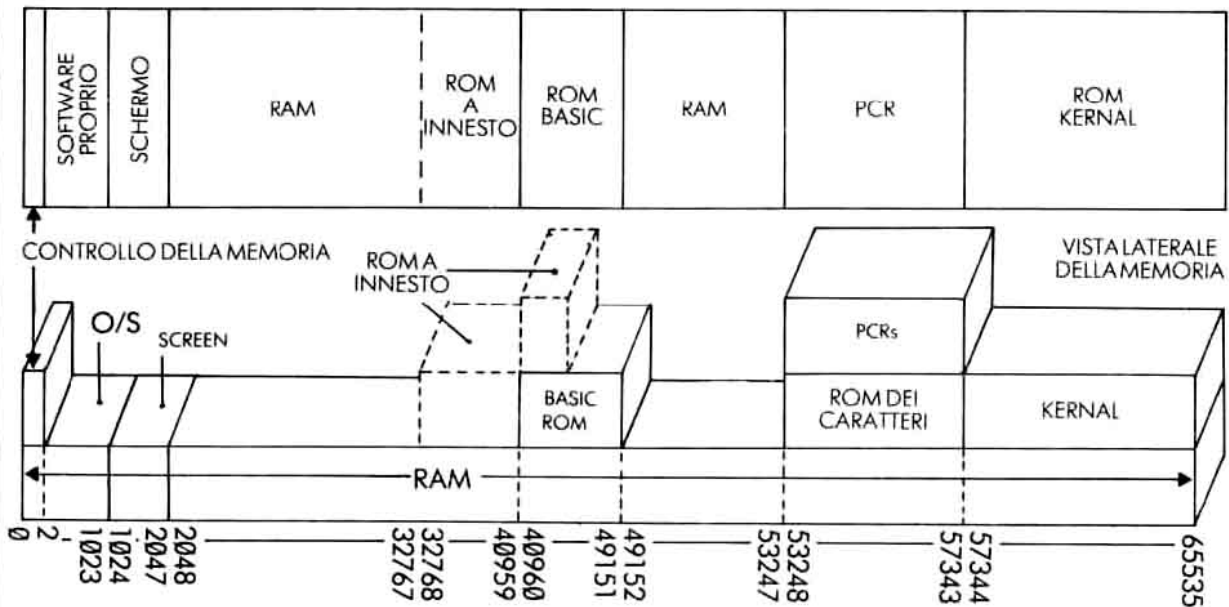



Figura 23.3

Mappa degli indirizzi per COMMODORE **64**

Come può il microprocessore avere accesso alla parte nascosta della memoria? Il segreto consiste nel registro all'indirizzo 1, che agisce all'incirca come un Registro di Controllo Periferiche. Ciascun bit nel registro controlla uno dei blocchi di memoria che ne nascondono altri. Tale blocco può essere fatto temporaneamente sparire rivelando un tipo diverso di memoria sottostante. Per esempio, il bit più a destra nel registro 1 controlla la ROM con l'interprete BASIC. Se si cambia questo bit da 1 a Ø, l'interprete scompare e il microprocessore avrà accesso diretto ai sottostanti 8K della RAM. Naturalmente non è possibile eseguire programmi BASIC quando la macchina è in questa condizione ma c'è altra RAM disponibile per i programmi scritti in altri linguaggi di programmazione tipo PASCAL e FORTH.
In modo analogo, il secondo bit più a destra rimuove il software KERNEL, che normalmente si occupa delle periferiche. Il terzo bit più a destra, consente al microprocessore di vedere la ROM dei caratteri, che è normalmente nascosta dai Registri di Controllo Periferiche.
Non c'è modo di rimuovere i Registri di Controllo Memoria veri e propri. Questi sono sempre disponibili agli indirizzi Ø e 1 e i byte Ø e 1 della RAM sono permanentemente coperti da essi.
In genere, solo i programmatori professionisti hanno necessità di spostare le ROM del BASIC e del KERNEL. In ogni caso, è spesso utile avere accesso alla ROM dei caratteri e di questo si parlerà più avanti nel corso di questa unità.

IL COMANDO "PEEK"

La funzione PEEK prende un indirizzo come argomento e ritorna i contenuti di quell'indirizzo sotto forma di un numero. Se l'indirizzo è in una parte della memoria in cui un tipo di memoria può nasconderne un'altro, il byte scelto è quello visibile al computer in quel momento.

È possibile applicare la funzione PEEK ad uno qualsiasi dei 65536 possibili indirizzi. Se si sceglie un'area visibile con la ROM (es. "6ØØØØ") è possibile trovare quale profilo era stato inserito in quel byte quando il computer è stato costruito.
Per avere un'altra rappresentazione del modo in cui i byte possono essere usati, si userà PEEK per dare uno sguardo alla ROM dei caratteri. Questa sezione di memoria contiene i *profili* dei vari caratteri così come compaiono sullo schermo.
La ROM dei caratteri è normalmente nascosta dai registri di controllo delle periferiche cosicché un modo comodo per esaminarla a piacere consiste nel copiarne i contenuti in una sezione di RAM che è sempre visibile al computer. Immettere ed eseguire il seguente programma, assicurandosi di batterlo correttamente prima di impartire il comando RUN:

```
1Ø A= 14336 : B = 53248 : C = 56334
2Ø POKE C, PEEK(C) AND 254
3Ø POKE 1, PEEK(1) AND 251
4Ø FOR J = Ø TO 2Ø47:POKE A+J,PEEK
   (B+J): NEXT J
5Ø POKE 1, PEEK(1) OR 4
6Ø POKE C, PEEK(C) OR 1
7Ø STOP
```

Questo programma copierà i primi 2Ø48 byte della ROM dei caratteri e li inserirà negli indirizzi da 14226 in su della RAM, svolgendo questo compito in circa 15 secondi.
I dettagli del programma potrebbero essere alquanto oscuri. Quando i registri di controllo periferiche vengono esclusi (cosa che deve succedere perché questo programma funzioni) essi devono essere per prima cosa disattivati o "disabilitati" altrimenti interferiscono con il programma e bloccano la macchina. La riga 2Ø nel programma ha il compito speciale di disabilitare i PCR.

La riga 3Ø altera il valore del registro 1 cosicché la ROM dei caratteri viene portata in vista perché il microprocessore la possa esaminare. La riga 4Ø sposta i byte e le righe 5Ø e 6Ø ripristinano lo stato originale cosicché è possibile eseguire di nuovo i programmi normali.
Quando si è eseguito il programma è possibile usare PEEK per ispezionare i contenuti dei byte da 14336 in su, che ora contengono una copia esatta (ma "visibile") della ROM dei caratteri. Iniziare con il seguente comando nel modo immediato (tutto su una riga):

```
FOR J=14344 TO 14351:PRINT J;PEEK(J):
NEXT J
```

Si ottiene:

```
14344  24
14245  6Ø
14346  1Ø2
14347  126
14348  1Ø2
14349  1Ø2
1435Ø  1Ø2
14351  Ø
```

Tutto ciò sembra abbastanza privo di significato ma vale la pena di osservare cosa succede se si convertono questi numeri in forma "binaria":

```
 24 : ØØØ11ØØØ        11
 6Ø : ØØ1111ØØ       1111
1Ø2 : Ø11ØØ11Ø      11  11
126 : Ø111111Ø      111111
1Ø2 : Ø11ØØ11Ø      11  11
1Ø2 : Ø11ØØ11Ø      11  11
1Ø2 : Ø11ØØ11Ø      11  11
 Ø  : ØØØØØØØØ
```

È ora chiaro il profilo della lettera "A". Quando il COMMODORE **64** visualizza una "A" sullo schermo, esso usa questi 8 byte dalla ROM per ottenere il profilo corretto della lettera.

ESPERIMENTO 23·1

(a) Usando PEEK, trovare quali caratteri sono memorizzati negli indirizzi da 14416 a 14423. Osservare a 145Ø3. Aiuta ciò a indovinare una relazione tra i caratteri e le loro posizioni nella ROM?

(b) Scrivere un programma che usa le subroutine che seguono per esporre la ROM dei caratteri e visualizzare versioni ingrandite di alcuni dei caratteri in essa contenuti.

```
1ØØØ REM DATO UN BYTE IN X1,
     TROVARE IL CORRISPONDENTE
     PROFILO BINARIO
1Ø1Ø REM E PRESENTARLO COME FILA
     DI ASTERISCHI E SPAZI
1Ø2Ø YY=256:FOR KK=1 TO 8
1Ø3Ø YY=YY/2
1Ø4Ø IF X1>=YY THEN X1=X1-YY:
     PRINT"★";:GOTO 1Ø6Ø
1Ø5Ø PRINT" ";
1Ø6Ø NEXT KK
1Ø7Ø PRINT:RETURN

15ØØ REM DATO UN INDIRIZZO IN Y1,
     INDICARE I CONTENUTI
151Ø REM NELLA ROM DEI CARATTERI
     IN X1
152Ø C=56334
153Ø POKE C, PEEK(C) AND 254:
     POKE 1, PEEK(1) AND 251
154Ø X1=PEEK(Y1)
155Ø POKE 1,PEEK(1) OR 4:POKE C,
     PEEK(C) OR 1
156Ø RETURN
```

Esperimento 23.1 completato	

IL COMANDO POKE

POKE è un vecchio amico, ma questa sezione ne illustra ulteriori aspetti.
Il comando POKE usa due argomenti: un indirizzo (nel campo da 0 a 65535) ed un numero compreso tra 0 e 255. L'effetto è di memorizzare il profilo binario di quel numero nell'indirizzo scelto. Per esempio,

POKE 54296,15

inserisce il profilo binario 00001111 nella locazione 54296. Naturalmente il comando funziona solo se l'indirizzo scelto è in una RAM o in un Registro di Controllo Periferiche. Se si esegue POKE in un indirizzo ROM (es. "6000"), il byte va nella RAM nascosta sottostante e non è possibile estrarlo di nuovo a meno che non si rimuova la ROM cambiando un bit nel registro di controllo di memoria. Un comando POKE può avere strani effetti se si sceglie un indirizzo RAM pertinente al KERNEL. Provare con il comando POKE 788,44: occorre in questo caso spegnere la macchina e riaccenderla per poterne riacquistare il controllo.
Un uso importante del comando POKE è il funzionamento dell'unità di controllo video, che è collegata a *quattro* diverse aree di memoria:

(a) I registri di controllo periferiche speciali (ad es. 53280 e 53281 che controllano i colori del riquadro e dello sfondo).

(b) La RAM dello schermo, tra gli indirizzi 1024 e 2047. Ciascun byte (salvo gli ultimi 24) contiene un codice per un carattere da visualizzare sullo schermo.

(c) La RAM dei colori, tra 55296 e 56295. Ciascun dei suoi byte indica il *colore* di un carattere da visualizzare.

(d) La ROM dei caratteri, che - si è già visto - contiene i *profili* dei vari caratteri.

L'immagine che si vede sullo schermo viene ridisegnata 50 volte ogni secondo e ogni volta il generatore cerca le locazioni 53280 e 53281 e dipinge i corrispondenti colori del riquadro e nel fondo. Quindi disegna ciascuno dei 1000 caratteri (uno "spazio" conta un carattere) iniziando dalla parte superiore sinistra e procedendo da sinistra a destra e dall'alto verso il basso.
Il processo usato per produrre ciascun carattere è abbastanza complesso. Ecco come si crea il primo carattere:
Il generatore inizia cercando nell'indirizzo 1024 che è la prima locazione nella RAM dello schermo. Qui trova un *codice di schermo* che dice quale è il carattere richiesto. Il codice è indicato in Fig. 23.4, dove per il momento di dovrebbe ignorare la colonna contrassegnata "SET2". Usando questo foglio di codifica, si vedrà che una "M" è rappresentata da 13 oppure un ♥ da 83.

Un codice di schermo può avere qualsiasi valore tra 0 e 255 ma la tabella dà soltanto i primi 127 in quanto quelli da 128 a 255 corrispondono agli stessi caratteri rappresentati in negativo. Per esempio, il codice per $ in negativo è 36+128 ossia 164.
Successivamente l'unità di controllo video cerca nella ROM dei caratteri il *profilo* da disegnare, moltiplicando il codice dello schermo per 8, aggiungendo 53248 ed estraendo 8 byte iniziando dall'indirizzo calcolato. Per esempio, ottiene la "M" dagli 8 byte che iniziano a 53248+13 ★ 8 ossia 53352. La ROM dei caratteri è sempre visibile all'unità di controllo video anche se è nascosta dal microprocessore 6510.
Infine, l'unità di controllo estrae dalla RAM dei colori il primo byte alla posizione 55296, ossia il *colore* del primo carattere secondo il codice standard:

0	1	2	3	4	5	6	7
Nero	Bianco	Rosso	Azzurro	Porpora	Verde	Blu	Giallo

8	9	10	11	12	13	14	15
Arancio	Marrone	Rosso chiaro	Grigio 1	Grigio 2	Verde chiaro	Azzurro	Grigio 3

L'unità di controllo video dispone ora delle informazioni per disegnare il primo carattere nel colore e nella forma corretti.
Disegnato il primo carattere, il generatore visualizza il secondo allo stesso modo, cercando però all'indirizzo 1025 il codice dello schermo ed al 55297 il colore.
L'unità di cotrollo video continua fino a che non ha disegnato tutti i 40 caratteri della prima fila, poi disegna quelli della seconda, terza e così via fino a riempire l'intero schermo. Quindi ricomincia da capo.
Il sistema sembra complicato ma è invece flessibile. L'unità di controllo funziona in continuazione ed abbastanza indipendentemente dal microprocessore. Per visualizzare qualsiasi informazione basta registrare i codici adatti nelle RAM dello schermo e dei colori. Ciò fatto, il nuovo carattere compare sullo schermo nel giro di 1/50 di secondo.
Si voglia visualizzare un rombo bianco alla 6ª riga, 14ª colonna. Ogni riga contiene 40 caratteri per cui questa posizione è visualizzata a 253 caratteri (6 ★ 40+13) dopo la posizione superiore sinistra estrema. La corrispondente cella nella RAM dello schermo è 1024+53=1277 e nella RAM dei colori è 5529+253=55549.
Dalla Fig. 23.4, il codice per un rombo è 90 e "bianco" è 1 per cui questa coppia di comandi dovrebbe inserire un rombo bianco al posto giusto:

POKE 1277,90:POKE 55549,1

CODICI DELLO SCHERMO

SET 1	SET 2	POKE
@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
[		27
£		28
]		29
↑		30
←		31
SPACE		32
!		33
"		34
#		35
$		36
%		37
&		38
'		39
(		40
)		41
*		42
+		43
,		44
—		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62

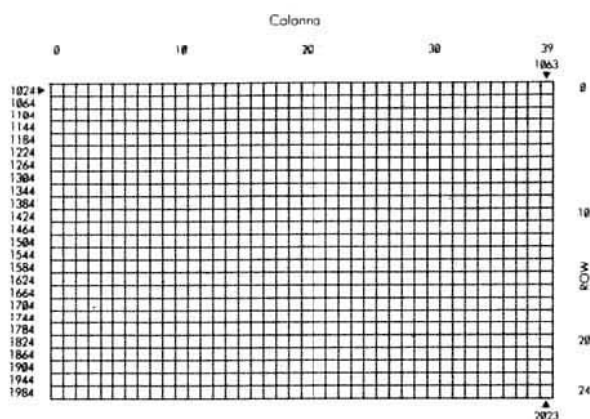
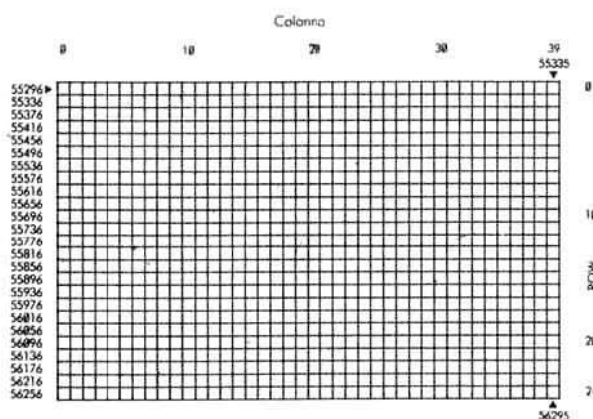
Figura 23.4

SET 1	SET 2	POKE
?		63
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
π		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
	√	122
		123
		124
		125
		126
		127

Figura 23.4 continua

I diagrammi in Fig. 23.5 aiuteranno a ricavare i corretti indirizzi RAM per qualsiasi punto dello schermo. È talvolta più comodo disegnare immagini usando i comandi POKE anziché i comandi PRINT. Ad esempio, ecco un programma che disegna una linea diagonale color porpora sullo schermo:

```
10 PRINT" SHIFT e CLR HOME "
20 POKE 53281,1:REM DIPINGI
   SCHERMO BIANCO
30 FOR J=32 TO 968 STEP 39
40 POKE 1024+J,78:POKE 55296+J,4
50 NEXT
60 GOTO 60:REM STOP ITERAZIONE
```

MAPPA DI MEMORIA DELLO SCHERMO

MAPPA DI MEMORIA DEI COLORI

ESPERIMENTO 23·2

Ecco un programma leggermente più lungo, MONDRIAN, che usa POKE per creare un effetto "artistico". Provarlo e cercare di migliorarlo, nei limiti del possibile.

```
   5 REM MONDRIAN (CON MOLTE
     SCUSE)
  10 PRINT " SHIFT e CLR HOME "
  20 FRO L=1 TO 8
  30 FOR K=0 TO 7
  40 POKE 53281,K+L
  50 FOR J=K TO 9
  60 Y1=2+J
  70 FOR X1=2+J TO 20-J
  80 GOSUB 1000
  90 NEXT X1
 100 X1=20-J
 110 FOR Y1=2+J TO 20-J
 120 GOSUB 1000
 130 NEXT Y1
 140 Y1=20-J
 150 FOR X1=20-J TO 2+J STEP-1
 160 GOSUB 1000
 170 NEXT X1
 180 X1=2+J
 190 FOR Y1=20-J TO 2+J STEP-1
 200 GOSUB 1000
 210 NEXT Y1
 220 NEXT J,K,L
 230 GOTO 230:REM STOP
     ITERAZIONE
1000 REM DISEGNA SPAZIO IN
     NEGATIVO IN X1+6,Y1+1.
     USA COLORE K
1010 ZZ=X1+40★Y1=46
1020 POKE 1024+ZZ,160:REM 160E
     CODICE SCHERMO PER SPAZIO
     IN NEGATIVO
1030 POKE 55296+ZZ,K
1040 RETURN
```

Esperimento 23.2 completato	

I CARATTERI NEL SET 2

Se si esamina la Fig. 23.4, si vedrà che le colonne contrassegnate SET 2 definiscono una serie diversa di caratteri. I caratteri mancanti nella colonna SET 2 sono identici ai corrispondenti nella colonna SET 1. In particolare, tutti i codici dello schermo che danno delle lettere maiuscole in SET 1 danno le stesse lettere minuscole in SET 2, mentre alcuni dei segni grafici in SET 1 sono sostituiti dalle lettere maiuscole. SET 2 è particolarmente adatto per i programmi di word processing.

Si seleziona SET 2 in due modi.

1. Se si sta corregendo un programma, abbassare SHIFT e premere [C=].
   Ripetere l'operazione per ritornare a SET 1.
2. Quando il programma richiede di usare SET 2, occorre impartire il comando

   POKE 53272,23

   Per ritornare a SET 1, occorre il comando

   POKE 53272,21

Notare che è difficile usare contemporaneamente caratteri da entrambi i set a meno che non si costruiscano proprie definizioni di caratteri. Ma ciò sarà spiegato nella prossima sezione.

COME DEFINIRE PROPRI CARATTERI

Si è visto come il COMMODORE **64** decide i profili dei vari caratteri che esso visualizza sullo schermo e si sa come copiare i byte che descrivono i caratteri nella RAM, dove essi sono accessibili. Il chip dell'unità di controllo video è flessibile al punto che gli si può dire di cercare i profili dei caratteri nella RAM anziché nella ROM dei caratteri. Da qui è breve il passo per disegnare e usare propri caratteri invece di quelli forniti dalla macchina.

La capacità di usare qualsiasi serie di caratteri aggiunge ulteriore flessibilità al **64**. Per esempio, è possibile visualizzare molte lingue diverse: russa, greca, araba, ebraica o malese. Inoltre è possibile definire caratteri per rappresentare mostri, navi spaziali e altri oggetti interessanti da usare nei giochi e nei programmi didattici.

Si comincia disegnando i caratteri che si intendono usare. Il processo che segue è facile anche se laborioso:

Disegnare un quadrato di 5 cm di lato e schizzare al suo interno la lettera desiderata. Stare staccati dai bordi a meno che non si desideri che il carattere tocchi quelli adiacenti. Tutti i segmenti dovrebbero essere spessi e in grossetto.

Successivamente, tracciare nel quadrato una griglia 8x8. Identificare le celle che cadono interamente o in gran parte all'interno delle parti nere del carattere e contrassegnarle con delle X. Assicurarsi che tutti i segmenti verticali e diagonali siano larghi almeno 2 celle.

Definire quindi ciascuna fila sotto forma di numero binario, inserendo degli 1 per le celle con le X e degli zeri per le celle vuote ed infine convertire ciascun numero nell'equivalente decimale.

La figura 23.6 mostra per esteso il processo necessario per copiare la lettera ("yat'").

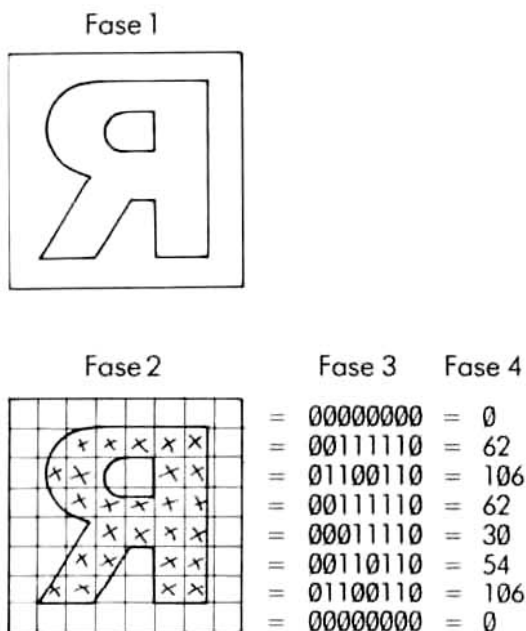



Figura 23.6

Una volta disegnati i nuovi caratteri, è possibile cominciare a lavorare sul programma che deve usarli. Nella maggior parte dei casi si useranno molti dei caratteri normali nonché quelli definiti per proprio conto. La descrizione che segue presuppone che la propria serie di caratteri sia basata su una di quelle normali (SET 1 o SET 2) leggermente modificate.

Iniziare studiando le serie di caratteri nella Fig. 23.4. Scegliere quella che piace di più quindi decidere quali caratteri vanno sostituiti con quelli prodotti "in proprio". Di solito si scartano alcuni dei simboli grafici di uso meno frequente o probabilmente le lettere in negativo (ricordarsi che i 128 simboli in negativo sono nello stesso ordine).

Per fare funzionare i nuovi caratteri, il programma deve procedere come segue:

1. Copiare una delle serie standard di caratteri dalla ROM di caratteri nelle locazioni da 14336 in avanti. Ricordare che ciascuna serie è lunga 2048 byte e che SET 1 inizia a 53248 mentre SET 2 inizia 2048 byte più in alto, vale a dire a 55296.

2. Sostituire le definizioni dei caratteri standard che non servono più con le descrizioni dei nuovi caratteri. Per far ciò, il programma deve conoscere:

   a) l'indirizzo del carattere da sostituire, e

   b) il profilo del nuovo carattere.

L'indirizzo di un carattere nella RAM è semplicemente 14336 più otto volte il codice dello schermo. Per esempio, l'indirizzo del ★ in SET 1 è 14336+8★42 ossia 14672. L'indirizzo di un carattere in negativo è 1Ø24 byte più elevato di quello del suo equivalente "normale".

Il profilo del nuovo carattere è dato dalla lista degli 8 numeri ricavati durante l'ultima fase del processo di disegno.
È comodo raccogliere tutte le informazioni sul nuovo carattere in un'unica istruzione DATA con 9 numeri: prima l'indirizzo del carattere da sostituire quindi gli 8 numeri che definiscono il nuovo profilo.
Per esempio, si potrebbe stabilire che la definizione della lettera russa Я sostituisca la cifra 1. Dato che il codice dello schermo di 1 è "49" e 14336+8★49=14728, l'appropriata istruzione DATA risulterebbe:

```
DATA 14728,Ø,62,1Ø6,62,3Ø,54,1Ø6,Ø
```

e un segmento di programma per impostare la nuova definizione potrebbe essere rappresentato dal seguente:

```
READ X:REM LEGGI INDIRIZZO
FOR J=ØTO 7
READ Y: REM LEGGI UN BYTE
        DI DESCRIZIONE
POKE X+J,Y
NEXT J
```

Quando sono state cambiate tutte le definizioni dei caratteri, la fase finale consiste nell'istruire l'unità di controllo video a cercare nella RAM i profili dei caratteri invece che cercarli nella ROM. Il comando per fare ciò è:

```
POKE 53272,31
```

Una volta cambiata la definizione di un carattere, *tutti* i riferimenti a quel carattere produrranno il nuovo profilo.
Come riprova, caricare ed eseguire il programma TONGUE, che ridefinisce le cifre 1,2,3 e 4 come caratteri russi Я, З, Ы e К quindi visualizzare la parola russa ЯЗЫК, che significa "TONGUE" o "LINGUA".

Notare che questo esempio va contro il nostro consiglio di sostituire soltanto "caratteri oscuri indesiderati". Ma c'è un buon motivo!

Provare a listare il programma (LIST) senza ripristinare il computer. Si vedrà che tutti gli 1, i 2, i 3 e i 4 compaiono ora come lettere russe! un RESTORE riporterà i vecchi caratteri al loro posto.
Questo programma contiene tutto ciò che serve per ridefinire e usare propri caratteri. Ne viene proposto un listato perché lo si possa studiare.

```
1Ø REM DEFINING YOUR OWN
   CHARACTERS
2Ø A=14336:B=53248:C=56334
3Ø POKE C,PEEK(C) AND 254: REM
   TRANSFER STANDARD SET OF
   CHARACTERS TO 14336
4Ø POKE1,PEEK(1) AND 251
5Ø FORJ=Ø TO 2Ø47:POKE(A+J),
   PEEK(B+J):NEXTJ
6Ø POKE 1,PEEK(1) OR 4
7Ø POKE C, PEEK(C) OR 1
8Ø REM
9Ø REM SET UP YOUR OWN CHARACTER
   DEFINITIONS
1ØØ REM EACH LINES HOLDS ADDRESS
    OF CHARACTER FOLLOWED BY 8
    VALUES WHICH DEFINE
11Ø REM ITS SHAPE. CHARACTERS ARE
    RUSSIAN LETTERS YA, Z, Y AND K.
12Ø REM AND THEY REPLACE DIGITS 1 2 3
    AND 4
13Ø DATA 14728,Ø,62,1Ø6,62,3Ø,54,1Ø6,Ø
14Ø DATA 14736,Ø,12Ø,6,12,12,6,12Ø,Ø
15Ø DATA 14744,Ø,199,199,243,223,223,
    243,Ø
16Ø DATA 14752,Ø,1Ø2,1Ø8,12Ø,12Ø,1Ø8,
    1Ø2,Ø
2ØØ FOR J=1 TO 4
21Ø READ X
22Ø FOR K= Ø TO 7
23Ø READ Y
24Ø POKE X+K,Y
25Ø NEXT K,J
26Ø REM NEXT SWITCH CHARACTER
    DESCRIPTION ADDRESS TO 14336
27Ø POKE 53272,31
28Ø REM NOW TRY IT OUT!
29Ø PRINT" CTRL and "2 SHIFT
    and CLR HOME CRSR ↑↓ 5 times CRSR ⇐⇒ 5 times
    1 2 3 4"
5ØØ STOP
```

ALTRO A PROPOSITO DEI PEEK E DEI POKE

Un modo comune per usare i PEEK consiste nell'esaminare cosa c'è sullo schermo. Se si dà ad un PEEK l'indirizzo di una cella nella RAM dello schermo, il risultato sarà il codice del carattere dello schermo in quella cella. Questa funzione è utile se si vuole disegnare un'immagine sullo schermo usando i comandi del colore e del cursore e quindi analizzare o registrare l'immagine con un programma.

Per consentire di disegnare l'immagine senza interferenze, scrivere comandi per cancellare lo schermo e immettere X (o qualsiasi altra variabile). Quando compare il punto di domanda, è possibile usare i comandi del cursore per disegnare qualsiasi immagine a piacere; è possibile anche cancellare il comando di input?.Per contro, non occorre usare RETURN fino a che l'immagine non è terminata; quando si preme RETURN il cursore deve essere in qualche posto in una riga completamente vuota, preferibilmente al disopra o al disotto dell'immagine.

Tutto ciò è illustrato nel seguente programma che conta e visualizza sullo schermo i numeri di caratteri diversi dallo spazio. Impostarlo, avviarlo, e quindi usare il controllo del cursore per distribuire alcuni simboli sullo schemo. Quindi battere RETURN.

```
10 INPUT"  SHIFT  e CLR HOME  ";X:REM X
   VARIABILE FITTIZIA
20 S=0
30 FOR J=0TO999:REM
   ANALIZZA SCHERMO
40 IF PEEK(1024=J)< >32THEN
   S=S+1:REM 32=CODICE
   SCHERMO PER SPAZIO
50 NEXT J
60 PRINT"NUMERO SIMBOLI=";S
70 STOP
```

Ci sono talune altre locazioni che possono essere comodamente usate con POKE per cambiare il comportamento del **64** in modi utili e prevedibili.

Per esempio, qui c'è un gruppo di locazioni che controllano il comportamento della tastiera.

★ Tasti con ripetizione: quando il **64** è acceso, i soli tasti che si ripetono se li si tiene abbassati sono lo spazio e i tasti di controllo del cursore. Ciò è controllato dal contenuto dell'indirizzo 650, che è interpretato come segue:

0: Si ripetono solo i tasti "standard"
127: Non c'è ripetizione di tasti
128: Tutti i tasti si ripetono

Se si impartisce il comando

POKE 650,128

si troverà che ora tutti i tasti si ripetono.

★ Coda della tastiera: se si battono caratteri più velocemente di quanto il programma non possa accettarli, essi verranno posti in una coda e forniti al programma uno alla volta. Il numero di caratteri nella coda può essere ispezionato eseguendo PEEK 198. I caratteri nella coda possono anche essere scartati inserendo (POKE) 0 nell'indirizzo 198. Ciò è spesso utile nei giochi dove lo scopo è di fare in modo che il computer reagisca a ciò che il giocatore sta facendo e non a ciò che ha fatto erroneamente qualche secondo prima.

UN ESEMPIO DI ANIMAZIONE

Terminiamo questa unità discutendo il disegno di un gioco animato. Caricare il programma intitolato "WASPS" ed eseguirlo parecchie volte fino a che non si è acquisita una certa esperienza come uccisore di vespe.

L'intero programma WASP è riprodotto alla fine di questa unità e ora daremo uno sguardo ai suoi principi generali di disegno.

Il gioco WASP, come la maggior parte degli altri giochi per computer, è una simulazione o se si preferisce una imitazione di qualcosa che si suppone succeda nel mondo esterno. In questo programma simula un cacciatore in una stanza piena di vespe. Le vespe si muovono a caso, mentre il cacciatore si muove, gira e lancia il suo insetticida spray in risposta ai comandi battuti sulla tastiera. Possono verificarsi vari eventi:

(a) Una vespa può essere uccisa

(b) Il cacciatore può essere punto

(c) Il cacciatore può finire l'insetticida spray.

Lo scheletro del programma è un modello, o una serie di variabili, che descrive completamente la posizione in qualsiasi istante. Una volta che questo modello è stato disegnato, è possibile scrivere vari pezzi di codice (prevalentemente subroutine) che operano sul modello e lo cambiano in conformità con gli eventi che si suppone accadano nel mondo esterno.

Ecco dunque una descrizione del modello del gioco WASP:

- N: Numero di vespe all'inizio
- NA: Numero di vespe rimaste in qualsiasi momento
- TT: Tempo di inizio della caccia (in jiffies)
- BU: Numero di colpi d'insetticida rimasti
- ST: Numero di volte che il cacciatore è rimasto punto
- XX$: Il numero di "vivi" che il cacciatore ha lasciato come una fila di tre ♥ . Questa riga è abbreviata di uno ogni volta che il cacciatore viene punto e quando non ci sono vivi il gioco si interrompe.
- IP: Altezza del ronzio della vespa (l'altezza del suono prodotto dalle vespe)
- A,B: La posizione attuale del cacciatore. A è il numero di colonna dello schermo da sinistra e B è il numero di righe dall'alto.
- C: L'attuale direzione del cacciatore:
  - 1=Nord
  - 2=Nord-Est
  - 3=Est
  - 4=Sud-Est
  - 5=Sud
  - 6=Sud-Ovest
  - 7=Ovest
  - 8=Nord-Ovest

La posizione di ciascuna vespa è registrata sotto forma di due elementi nella matrice W%(N,2). Pertanto la posizione di colonna della prima vespa è W%(1,1) e la sua posizione di riga è W%(1,2). La seconda vespa occupa W%(2,1) e W%(2,2) e così via.
La posizione dell'ultima vespa attiva è memorizzata in W%(NA,1) e W%(NA,2). Se una vespa (diversa dall'ultima vespa attiva) viene uccisa, il record di tutte quelle al di sotto di essa viene spostato verso l'alto per riempire lo spazio rimasto vuoto.

Per esempio:

NA=6

W%

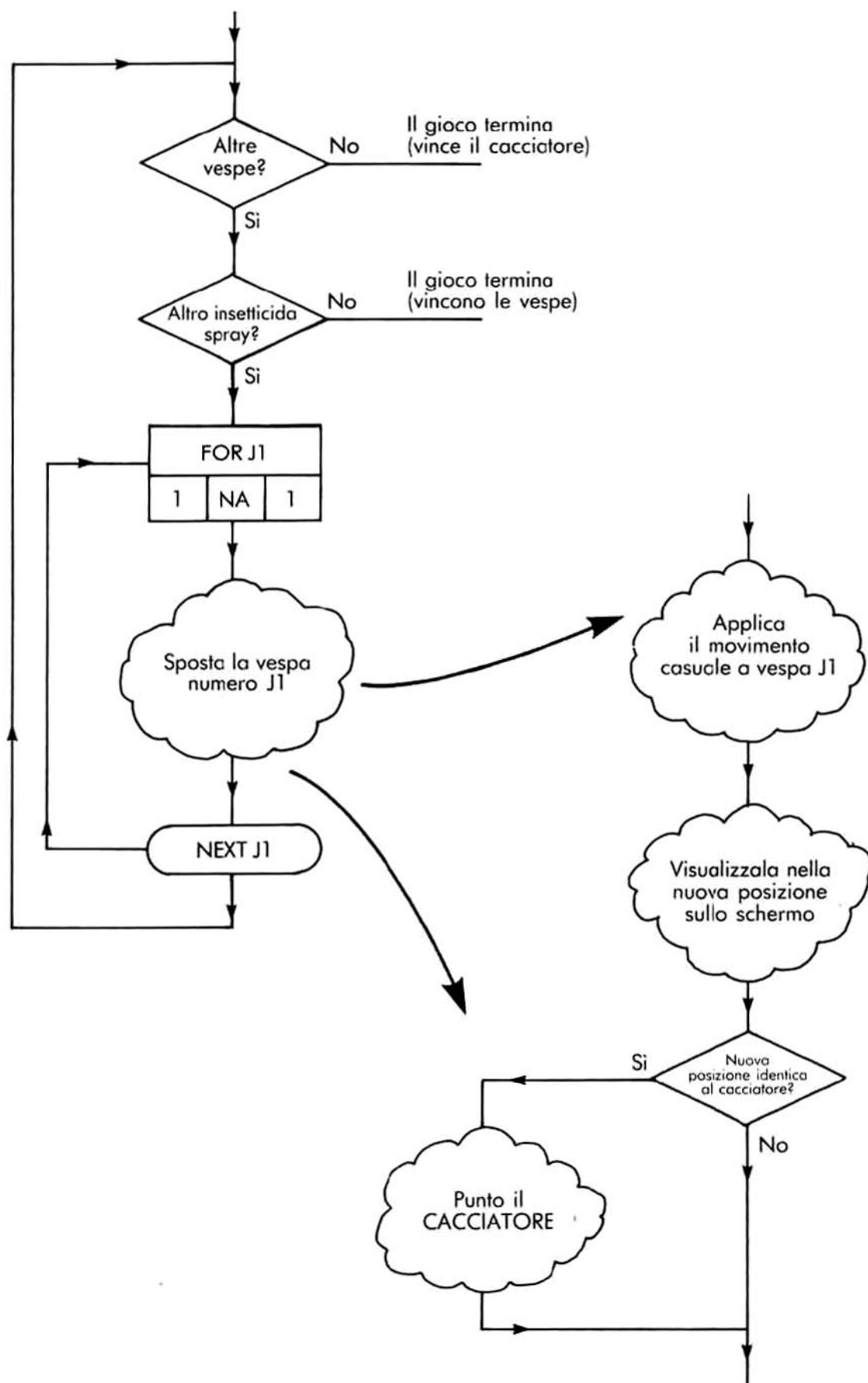
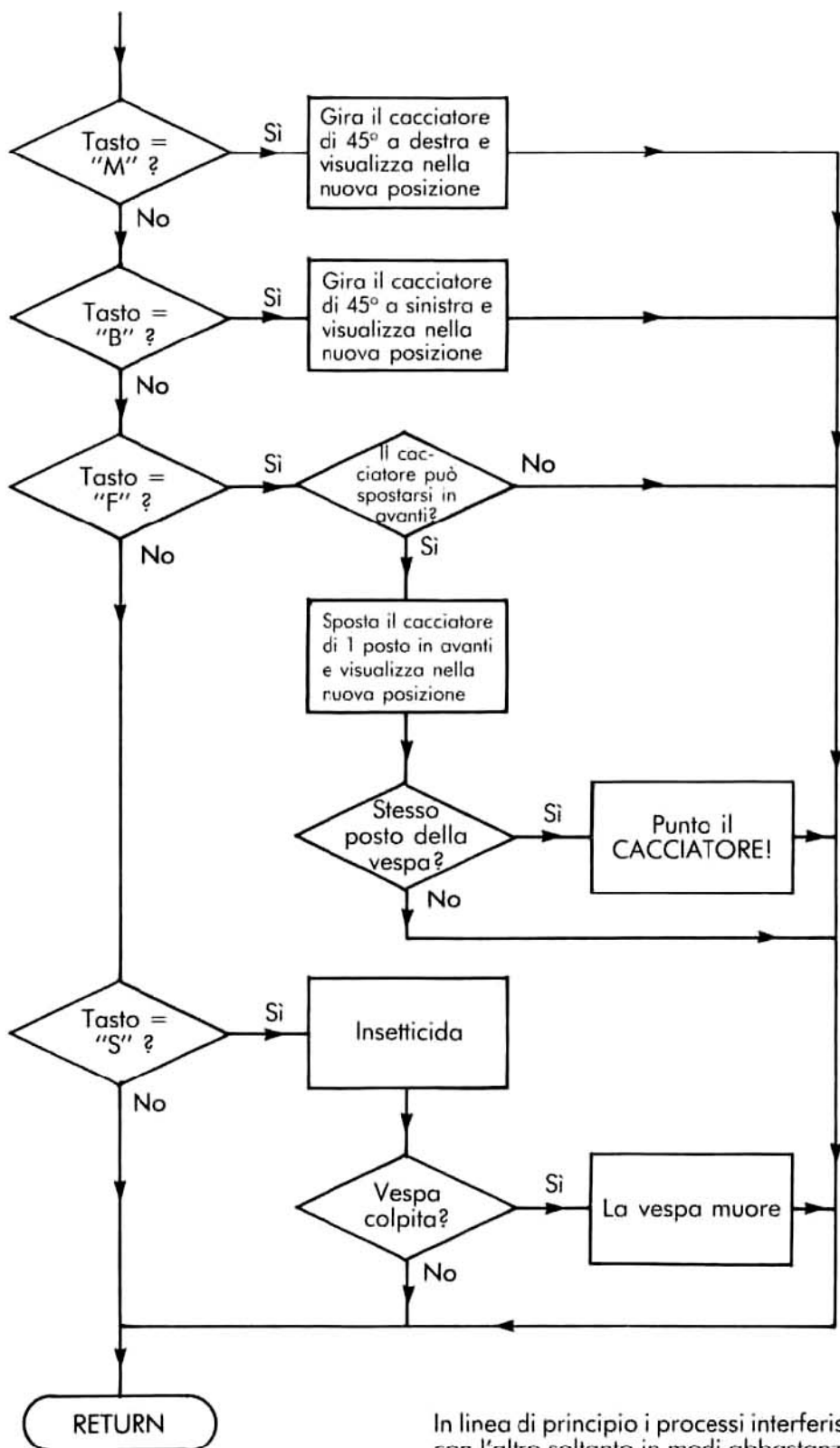
1	17	
3	12	
18	2	
10	7	← Questa è uccisa
7	9	
4	17	

dà

NA=5

W%

1	17	
3	12	
18	2	
7	9	
4	17	
—	—	Questa fila è inutilizzata

Il gioco stesso è organizzato sotto forma di due processi che vengono eseguiti pressochè indipendentemente: il processo "vespa" e il processo "cacciatore".
Il processo "vespa" è responsabile di muovere tutte le vespe. Una per una, ciascuna vespa nella lista viene spostata casualmente al massimo di una cella, con la limitazione che non deve cadere fuori dal margine dello schermo. Se una vespa si muove nello stesso posto del cacciatore, lo punge.
Il processo vespa viene eseguito ripetutamente fino a che o non rimangono più vespe oppure non si esaurisce l'insetticida. Il suo schema di flusso è:

Il processo "cacciatore" è richiamato ogniqualvolta viene premuto un tasto. Il suo schema di flusso è:

In linea di principio i processi interferiscono l'uno con l'altro soltanto in modi abbastanza specifici. Per esempio, le attività del cacciatore riducono gradualmente il numero di vespe attive e la quantità di insetticida rimanente, al limite provocando l'interruzione del processo.
In pratica, occorre fare in modo che entrambi i processi si svolgano contemporaneamente (più o meno). Un modo semplice per far ciò è di dare al cacciatore una possibilità di fare qualcosa dopo che ciascuna vespa si è mossa. Viene estratto un carattere dalla tastiera e se viene premuto un tasto, è possibile dare inizio al "processo del cacciatore".

È possibile ora dare una descrizione dettagliata del programma.

La riga 10 dichiara quattro matrici. Sono tutte matrici intere, dato che si guadagna spazio in memoria. Come si sa, W% è usato per accogliere la posizione di ciascuna vespa attiva. V%, U% e D% aiutano tutte a visualizzare e a spostare il cacciatore come verrà spiegato più avanti.

Le righe da 20 a 90 visualizzano una serie di istruzioni dell'utente.

Le righe da 110 a 118 impostano taluni numeri in forma simbolica. Ciò renderà più facile la successiva lettura del programma. Per esempio, PA è l'indirizzo iniziale del generatore di suoni e un indirizzo tipo "PA+19" è più comprensibile e certamente più corretto di "54291".
Queste righe inizializzano anche il generatore di suoni, selezionano il nero come colore di fondo e trasformano tutti i tasti della tastiera in "ripetitori".

Le righe da 120 a 130 determinano il numero di vespe con cui iniziare.

Le righe da 150 a 210 impostano i valori appropriati per le matrici V%, U% e D%.

Si ricorderà che i quadrati nella RAM dello schermo sono numerati da sinistra a destra iniziando con la fila superiore e procedendo verso il basso.

		1256		
	1295	1296	1297	1298
1334	1335	1336	1337	1338
1374	1375	1376	1377	
		1416		

Qui c'è una parte dello schermo dove ciascuna cella è contrassegnata con il proprio indirizzo. Si supponga che il cacciatore si trovi in qualche cella con la mira puntata a Nord. Come si può vedere, il contenitore di insetticida occuperà il quadrato avente il numero 40 meno quello occupato da se stesso. Analogamente, se il cacciatore punta a Nord-Est, la cella di mira avrà un numero inferiore di 39 e così via. La matrice V% contiene la "differenza del numero di cella" per ciascuna delle 8 possibili direzioni, iniziando con V%(1) (Nord) e procedendo fino a V%(8) (Nord-Est). In effetti, (tenendo presente che la variabile C contiene la direzione verso la quale è rivolto il cacciatore, l'appropriata differenza del numero di cella è sempre V%(C).
La matrice U% contiene i codici dello schermo dei simboli usati per rappresentare la mira del cacciatore. Essi variano secondo la direzione:

N	NE	E	SE	S	SW	W	NW
|	/	—	\	|	/	—	\

Usando queste tabelle il cacciatore può facilmente venir visualizzato in qualsiasi posizione e in qualsiasi direzione. Osservare le righe 2000-2020 ricordando che la posizione del cacciatore è registrata nelle variabili A e B. SR è la costante 1024 che è l'indirizzo della prima cella nella RAM dello schermo.
La matrice D% mostra i movimenti EST e SUD che corrispondono ad una mossa in una qualsiasi delle 8 direzioni.
I valori sono:

N	0	−1
NE	1	−1
E	1	0
SE	1	1
S	0	1
SW	−1	1
W	−1	0
NW	−1	−1

Ciò rende facile spostare il cacciatore. Per esempio, per andare in un quadrato in direzione 6 (Sud-Ovest), si aggiunge D%(6,1) ad A e D%(6,2) a B.

La riga 220 imposta il rifornimento iniziale di insetticida. La formula prevede una tabella di questo genere.

N. delle vespe	1	2	3	4	5	6	7	8
N. degli spruzzi	7	9	12	14	15	17	18	19

Ciò tiene conto del fatto che le prime vespe, quando ne sono presenti molte, sono molto facili da colpire che non l'ultima.

Le righe 250 a 290 elaborano le posizioni iniziali per tutte le vespe e le visualizzano sullo schermo. Innanzitutto tutte le vespe sono inserite nella metà superiore dello schermo (righe da 1 a 12). Notare (280) il secondo POKE per impostare il colore della vespa.

La riga 320 fissa la posizione iniziale del cacciatore e lo visualizza in questa posizione.

Le righe da 330 a 420 formano il cuore della simulazione; esse eseguono il processo del cacciatore ogniqualvolta viene premuto un tasto. 340 e 350 visualizzano la situazione corrente. Notare che la subroutine in 1000 muove la vespa J1 e quella in 3000 attiva il cacciatore.

Le righe da 500 a 570 visualizzano i messaggi finali di congratulazioni o di consolazione, a secondo dei casi.
La subroutine che muove le vespe è alle righe da 1000 a 1090. Il metodo base è da ottenere la posizione precedente della vespa nelle variabili locali XN e YN. Ciascuno di questi numeri viene quindi "perturbato" da una quantità casuale che può essere +1, 0 o -1.
Se il risultato è al di fuori del campo dello schermo è respinto e il processo viene ripetuto.
Quando è stata determinata una nuova posizione, la vespa nella vecchia posizione viene cancellata (riga 1040). Viene disegnata una vespa nella nuova posizione a meno che questa non sia già occupata.
Quindi la nuova posizione viene registrata nella tabella in W%(J1,1) e W% (J1,2).
È inclusa la prova in 1045 in quanto la subroutine per il movimento delle vespe non deve poter spostare le vespe uccise di recente.
A questo punto viene generato il ronzio della vespa. L'altezza corrente del ronzio è memorizzata nella variabile IP. Il valore IP è perturbato da un'unità casuale, con una trappola per impedirgli di allontanarsi troppo dal suo campo normale di circa 70. Quindi la variabile viene usata per iniziare una nota ronzante che continua fino a quando non viene cambiato IP.
Infine, la nuova posizione della vespa è confrontata con quella del cacciatore e se corrispondono, viene richiamata la subroutine di "puntura" a 4000.

Le righe da 2000 a 2020 visualizzano il cacciatore nella sua nuova posizione, e

Le righe da 2500 a 2520 cancellano il cacciatore da una vecchia posizione disegnando degli spazi nei punti appropriati.

Le righe da 3000 a 3270 sorvegliano il cacciatore. Le parti di questa subroutine sono le seguenti: da 3020 a 3030 rotazione verso destra. Notare che il Nord (C=1) deve seguire Nord-Est (C=8).
Da 3050 a 3060 rotazione a sinistra. Notare che Nord-Est (C=8) deve seguire Nord (C=1).
Da 3080 a 3140 spostamento in avanti. Viene prodotta una nuova posizione orientativa in AA e BB ed è trasferita soltanto a A e B se non è troppo vicina al margine dello schermo. Quando viene fatta la mossa viene esaminata la lista delle posizioni delle vespe per vedere se il cacciatore si è seduto su una vespa; in questo caso è punto. Ciò accade nelle righe da 3110 a 3130.
Da 3160 a 3270 spruzzo. PP e QQ sono le posizioni dell'area bersaglio. Le righe 3170 e da 3190 a 3240 sono interessate agli effetti (suono e visione) dello spruzzo. Le righe da 3250 a 3270 esaminano la lista delle vespe per vedere se ne è stata colpita una. In questo caso viene richiamata la subroutine in 5000.

Le righe da 4000 a 4140 rappresentano le subroutine richiamata quando il cacciatore viene punto, che è principalmente costituita da effetti audiovisivi anche se c'è la possibilità che il cacciatore salti ad una nuova posizione casuale oppure di terminare il gioco se sono state esaurite le vespe vive a disposizione del cacciatore.

Le righe da 5000 a 5100 riguardano la morte della vespa. A parte gli effetti soliti, il record della vespa morta è rimosso dalla lista e i record vengono spostati verso l'alto, se necessario.

```
1 REM WASPSHOOTER COPYRIGHT
  (C) ANDREW COLIN 1983

10 DIMV%(8),U%(8),D%(8,2),W%(20,2)
20 PRINT" CTRL and 2 SHIFT
   and CLR HOME CRSR
   WASPSHOOTER":PRINT:PRINT
25 PRINT"COPYRIGHT (C) ANDREW
   COLIN 1983"
28 PRINT
30 PRINT"KILL ALL THE WASPS"
40 PRINT"BEFORE THE FLY-SPRAY"
50 PRINT "RUNS OUT"
60 PRINT: PRINT" M TO TURN RIGHT"
70 PRINT" B TO TURN LEFT"
80 PRINT" F TO GO FORWARD"
90 PRINT" S TO SHOOT":PRINT
95 XX$=" CTRL and 3
   SHIFT and CLR HOME 3 times"
100 SR=1024:PA=54272:FORJ=0TO23:
    POKEPA+J,0:NEXTJ
110 POKE PA+2,100:POKEPA+5,15:
    POKEPA+24,15:POKE PA+8,100:
    POKE PA+12,15:IP=70
115 POKE PA+19,15:POKEPA+6,224:
    POKEPA+4,0:POKEPA+4,65
118 POKE 53281,0:POKE 650,128
120 INPUT"HOW MANY WASPS";N
130 IF N<1 OR N>20 THEN PRINT "1 TO
    20 PLEASE":GOTO 120
150 FOR J = 1 TO 8
160 READ V%(J),U%(J),D%(J,1),D%(J,2)
170 NEXT J
180 DATA -40,93,0,-1,-39,78,1,-1
190 DATA 1,67,1,0,41,77,1,1
200 DATA 40,93,0,1,39,78,-1,1
210 DATA -1,67,-1,0,-41,77,-1,-1
220 BU=INT(7★SQR(N)):SQ=0
230 PRINT" SHIFT and CLR HOME "
240 FOR J=55296 TO 56295 :
    POKEJ,7:NEXT J
250 FORJ=1TON
260 W%(J,1)=INT(40★RND(0))
270 W%(J,2)=INT(12★RND(0))+1
280 POKE SR+40★W%(J,2)+W%(J,1),35
290 NEXTJ
300 NA=N
310 TS=TI
320 A=3:B=18:C=2:GOSUB2000
330 IFNA=0THEN500
335 IFBU=0THEN600
340 PRINT" CTRL and 8 CLR HOME "
```

```
350 PRINT" [CLR HOME] WASPS";NA;"TIME";
     INT((TI-TS)/60);"SHOTS";BU;
     "LIVES";XX$
370 FORJ1=1TONA
380 IFW%(J1,1)>=0THENGOSUB1000
390 GETA$:IFA$=" "THEN410
395 POKE 198,0
400 GOSUB3000
410 NEXTJ1
420 GOSUB2000:GOTO330
500 REM WINS
510 PRINT" [SHIFT] and [CLR HOME] [CRSR ↑↓]
     [CRSR ↑↓] WELL DONE !":PRINT
520 PRINT" YOU HAVE KILLED ";N-NA:
     PRINT
530 PRINT"WASPS IN"INT((TI-TS)/60);
     "SECONDS":PRINT
540 PRINT"YOU WERE STUNG":PRINT
550 PRINTSQ;" TIMES"
560 POKEPA+24,0:RESTORE
565 FOR TT=1 TO 5000:NEXT TT
570 PRINT:PRINT" TO HAVE ANOTHER
     GAME HIT ANY KEY"
580 GETA$:IFA$=" " THEN 580
590 GOTO 20
600 REM OUT OF FLY-SPRAY
610 PRINT" [SHIFT] and [CLR HOME] [CRSR ↑↓]
     [CRSR ↑↓] SORRY—NO FLY-SPRAY":PRINT
620 PRINT"LEFT!":PRINT
630 GOTO520
1000 REM MOVE J1'TH WASP AT RANDOM
1010 XX=W%(J1,1):YY=W%(J1,2)
1020 XN=XX+INT(3*RND(0))-1:IFXN<0
      OR XN>39 THEN 1020
1030 YN=YY+INT(3*RND(0))-1:IF YN<1
      OR YN>24 THEN GOTO1030
1040 POKESR+40*YY+XX, 32
1045 IF J1>NA THEN 1070
1050 ZZ=SR+40*YN+XN:IFPEEK(ZZ)=32
      THEN POKE ZZ,35
1060 W%(J1,1)=XN:W%(J1,2)=YN
1070 IP=IP+INT(3*RND(0))-1: IF IP<60
      OR IP>80 THEN IP=70
1080 IFXN=A AND YN=BTHENGOSUB
      4000
1090 POKEPA,64*(IP AND 3):POKE
      PA+1,IP/4:RETURN
2000 REM DISPLAY HUNTER
2010 XX=SR+40*B+A:YY=XX+V%(C)
2020 POKEXX,81:POKEYY,U%(C):RETURN
2500 REM ERASE HUNTER
2510 XX=SR+40*B+A:YY=XX+V%(C)
2520 POKEXX,32:POKEYY,32:RETURN
3000 REM MOVE HUNTER OR SHOOT
3010 IFA$<>"M"THEN3040
3020 GOSUB2500:C=C+1:IFC=9THEN
      C=1
3030 GOSUB2000:RETURN
3040 IFA$<>"B"THEN3070
3050 GOSUB2500:C=C-1:IFC=0THEN
      C=8
3060 GOSUB2000:RETURN
3070 IFA$<>"F"THEN 3150
3080 GOSUB2500:AA=A+D%(C,1):
      BB=B+D%(C,2)
3090 IF AA>2 AND AA<36 AND BB>2
      AND BB<22 THEN A=AA:B=BB
3100 GOSUB2000
3110 FORJJ=1TONA
3120 IFA=W%(JJ,1)ANDB=W%(JJ,2)
      THENGOSUB4000
3130 NEXTJJ
3140 RETURN
3150 IFA$<>"S"THEN RETURN
3160 PP=A+2*D%(C,1):QQ=B+2*D%
      (C,2)
3170 POKEPA+11,129
3180 BU=BU-1
3190 RR=SR+40*QQ+PP
3200 FORKK=1TO5
3210 POKERR,102:FORTT=1TO30:
      NEXTTT
3220 POKERR,32:FORTT=1TO50:
      NEXTTT
3230 NEXTKK
3240 POKEPA+11,0
3250 FORJJ=1TONA
3260 IFPP=W%(JJ,1) ANDQQ=W%(JJ,2)
      THENJ2=JJ:GOSUB5000
3270 NEXTJJ:RETURN
4000 REM HUNTER IS STUNG
4010 PRINT" [CTRL] and [( 8] [CLR HOME]
      S T U N G ! ! ! "
4015 XX$=LEFT$(XX$,LEN(XX$)-1)
4020 GOSUB2500
4030 A=INT(3+16*RND(0)):
      B=INT(3+16*RND(0)):
      C=INT(1+8*RND(0))
4040 POKEPD,15:GOSUB2000:
      SQ=SQ+1
4050 POKEPA+18,17
4060 FORJJ=128 TO 255 STEP 3
4070 POKE PA+15,256-JJ:POKE 53281,
      JJ:POKE 53280,JJ-1
4080 NEXT JJ
4085 IF LEN(XX$)=1 THEN 4100:REM JUMP
      IF LIVES USED UP
4090 POKE PA+18,0:POKE 53281,0:
      RETURN
4100 POKE PA+24,0
4110 PRINT" [SHIFT] and [CLR HOME] [CRSR ↑↓] [CRSR ↑↓]
      [CRSR ↑↓] [CRSR ⇐⇒] [CRSR ⇐⇒] [CRSR ⇐⇒] [CTRL] and
      [" 2] YOU HAVE BEEN STUNG
      THREE TIMES"
4120 PRINT " [CRSR ↑↓] [CRSR ↑↓] [CRSR ⇐⇒] [CRSR ⇐⇒] [CRSR ⇐⇒]
      AND YOUR CONSTITUTION CAN
      NO LONGER"
4130 PRINT " [CRSR ↑↓] [CRSR ↑↓] [CRSR ⇐⇒] [CRSR ⇐⇒] [CRSR ⇐⇒]
      STAND IT"
4135 PRINT
4140 GOTO520
```

```
5000 REM WASP IS KILLED
5010 PRINT" [CTRL] and [2] [CLR HOME] A
     WASP BITES THE DUST!"
5020 FOR JJ=4 TO 92 STEP 4: POKE
     PA+15,100-JJ
5030 POKE PA+18,17
5040 FORTT=1TO10:NEXTTT
5050 POKE PA+18,0:FORTT=1TO10:
     NEXTTT
5060 NEXTJJ
5070 IFJ2=NATHENNA=NA-1:RETURN
5080 W%(J2,1)=W%(J2+1,1)
5090 W%(J2,2)=W%(J2+1,2)
5100 J2=J2+1: GOTO 5070
```


ESPERIMENTO 23·3

Disegnare e programmare un proprio gioco animato. Provare, ad esempio, a:

Colpire alieni provenienti dallo spazio esterno.

Ricercare la strada in un labirinto essendo contemporaneamente inseguiti da un mostro.

Catturare palle lanciate casualmente.

Esperimento 23.3 completato	

UNITA':24

Altro sugli operatori logici	PAGINA 253
Come il 64 valuta le condizioni	253
Codici ASCII del CBM	254
Uso di ASC-Conteggio della comparsa delle lettere	256
Uso di ASC-Come ignorare l'imput illecito	257
Esperimento 24.1	259
Il comando "ON"	260
Il comando "END"	260
Il comando "DEF"	260
Esperimento 24.2	261
Memorizzazione e richiamo di dati	262
Dati su cassetta	262
PRINT # per scrivere i dati	262
INPUT # per leggere i dati	263
GET #	264
Problemi	264
Dati su dischetti	265
Il comando "OPEN"	266
PRINT # per scrivere i dati	266
INPUT # per leggere i dati	267
Esperimento 24.3	268
Esperimento 24.4	269

ALTRO A PROPOSITO DEGLI OPERATORI LOGICI

In questa unità completeremo il nostro studio del **64** BASIC considerando alcuni argomenti vari.
L'Unità 17 esaminava l'uso degli operatori logici AND, OR e NOT usati per costruire condizioni composte. Gli stessi operatori possono anche essere usati in un contesto completamente diverso per manipolare le cifre binarie in numeri e in altre variabili.
Battere il comando

```
PRINT 13 AND 17
```

Il risultato, 5, è completamente misterioso fino a che non si osserva la rappresentazione binaria dei numeri coinvolti:

```
13 = ....0001101
 7 = ....0000111
 5 = ....0000101
```

Come è possibile vedere, il risultato ha un "1" soltanto nelle colonne in cui entrambi i numeri originali avevano "1". Possiamo spiegare l'operazione AND con una "tavola di verità" che si applica indipendentemente a ciascuna colonna:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Usando questa tavola, è possibile prevedere il risultato del comando

```
PRINT 27 AND 6
```

```
27 = ...0011011
 6 = ...0000110

        0000010=2
```

Per assicurarsi di aver compreso il funzionamento di AND, provare a calcolare in anticipo i risultati di quanto segue:

```
PRINT 15 AND 12
PRINT 21 AND 10
PRINT 11 AND 7
```

Controllare il calcolo sul **64** in ciascun caso.
L'operatore OR è molto simile a AND con la differenza che dà un "1" in qualsiasi colonna dove uno o l'altro o entrambi i numeri originali erano "1". La sua tavola di verità è:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

Usando questa tavola non si dovrebbero avere problemi nel prevedere il risultato del comando.

```
PRINT 7 OR 10
```

L'operatore NOT si limita a prendere un singolo numero e a cambiare ogni suo bit in quello contrario. Si troverà che

```
NOT (....0001010) = ....1110101
```

Nel **64** (e in conseguenza nella maggior parte degli altri computer) un numero che si compone interamente di uni rappresenta −1 (1 negativo). Come ci si sarebbe aspettato, il risultato del comando

```
PRINT NOT 0
```

è −1.
Le operazioni AND, OR e NOT sono utili nel lavorare con quantità dove le singole cifre binarie hanno significati speciali. Per esempio, se si usa il **64** per controllare le luci di casa attraverso la porta utente (user port), è abbastanza probabile che le 8 posizioni dei singoli interruttori siano rappresentate come le 8 cifre binarie di un singolo numero. Per scoprire se, ad esempio, il quinto interruttore da destra è acceso, occorrerebbe eseguire un'operazione AND tra il carattere e il numero binario . .0010000. Il risultato sarebbe diverso da 0 soltanto se il numero avesse un "1" in quella posizione — in altre parole, se il quinto interruttore fosse acceso.
L'equivalente di . .0010000 è 16 cosicchè il programma di controllo potrebbe contenere una riga del tipo

```
360 IF (S AND 16)<>0 THEN 590
```

COME IL **64** VALUTA LE CONDIZIONI

Ci si potrebbe domandare come questi apparenti nuovi significati degli operatori si colleghino con quelli convenzionali usati nelle condizioni composte. Per scoprire la risposta occorre osservare un po' più in profondità i meccanismi del **64**. Quando la macchina elabora una semplice condizione (ad esempio X=5 o A$<>"YES" o 5=4), essa produce sempre un valore di verità che è −1 se la condizione è vera e 0 se è falsa. Provare i seguenti comandi (quantunque sembrino strani) e spiegare perchè essi producano i risultati che in effetti producono:

```
PRINT 5=4
PRINT 6<9
PRINT 9>6
PRINT (1=1)★(1<2)
```

Il comando IF comprende sempre un'espressione tra IF e THEN. Questa è solitamente una condizione, ma non deve esserlo necessariamente; forme del tipo

```
IF X −3 THEN
```

sono abbastanza accettabili. Il comando (o gruppo di comandi) che segue THEN è eseguito se l'espressione dopo IF ha un qualsiasi valore salvo lo 0. Eseguire il seguente programma e spiegarne i risultati:

```
10 FOR X = 1 TO 5
20 IF X -3 THEN PRINT X
30 NEXT X
40 STOP
```

Anche quando l'espressione è una condizione, il modo in cui la macchina lavora è sempre lo stesso. Si consideri il comando

```
IF "PAPERINO"<"TOPOLINO" THEN
PRINT "PLUTO"
```

Si vedrà che la condizione è vera e ci si aspetta che la macchina visualizzi di conseguenza la stringa "PLUTO". Il **64** passa in effetti attraverso una fase intermedia. Esso per prima cosa valuta la condizione a "1" quindi, esegue il comando PRINT in quanto −1 non è lo stesso di 0.
Per completare la spiegazione delle condizioni composte, tutto ciò che si deve dire è che gli operatori logici possono essere applicati ai valori di verità prodotti da condizioni semplici. Si supponga che X$="D". Di conseguenza la condizione composta all'interno dell'espressione

```
IF NOT(X$="C" OR X$="D")
```

si traduce in

```
  NOT (0 OR -1)
= NOT (....00000 OR ....11111)
= NOT (....11111)
= ....00000
```

per cui la condizione è falsa.

I CODICI ASCII DEL CBM

Il prossimo argomento è la rappresentazione interna dei caratteri. Si sa già che una stringa, quando memorizzata internamente, richiede un byte per ciascun carattere che essa contiene. È talvolta utile conoscere esattamente come viene rappresentato ciascun carattere.
Nell'Unità 23 abbiamo introdotto l'idea di un "codice dello schermo" e abbiamo dato una tabella che dimostrava come ciascun carattere che poteva essere visualizzato sullo schermo, avesse un proprio codice speciale. Anche all'interno del **64** i caratteri sono rappresentati da un codice, ma questo è diverso dal codice dello schermo! È possibile vedere che deve essere diverso in quanto il codice deve essere in grado di gestire ogni carattere prodotto premento un tasto sulla tastiera. Ciò comprende tasti tipo RETURN o i movimenti del cursore che non corrispondono a qualsiasi simbolo visualizzabile.
Il codice usato è una modifica dell'American Standard Code for Information Interchange (abbreviato, "ASCII"). Questo codice consente di trasmette informazioni su linee telefoniche tra macchine di vari tipi.

Key	1	2	3	4
←	95	95	95	6
1	49	33	33	144
2	50	34	34	5
3	51	35	35	28
4	52	36	36	159
5	53	37	37	156
6	54	38	38	30
7	55	39	39	31
8	56	40	40	158
9	57	41	41	18
Ø	48	48	48	146
+	43	219	166	—
−	45	221	22Ø	—
£	92	169	168	—
CLR HOME	19	147	147	—
INST DEL	20	148	148	—
F1/F2	133	137	137	—
CTRL				
Q	81	209	171	—
W	87	215	179	—
E	69	197	177	—
R	22	21Ø	178	18
T	84	212	163	—
Y	89	217	183	—
U	85	213	184	—
I	73	2Ø1	162	—
O	79	2Ø7	185	—
P	8Ø	2Ø8	175	—
@	64	186	164	—
★	42	192	223	—
↑	94	222	222	—
RESTORE				
F3/F4	134	138	138	—
RUN STOP				
SHIFT LOCK				
A	65	193	176	—
S	83	211	174	—
D	68	196	172	—
F	7Ø	198	187	—
G	71	199	165	—
H	72	2ØØ	18Ø	—
J	74	2Ø2	181	—
K	75	2Ø3	161	—
L	76	2Ø4	182	—
:	58	91	91	—
;	59	93	93	—
=	61	61	61	—
RETURN	13	141	141	—
F5/F6	135	139	139	—
C=				
SHIFT				
Z	9Ø	218	173	—
X	88	216	189	—
C	67	195	188	—
V	86	214	19Ø	—
B	66	194	191	—
N	78	2Ø6	17Ø	—
M	77	2Ø5	167	—
,	44	6Ø	6Ø	—
.	46	62	62	—
/	47	63	63	—
SHIFT				
CRSR ⇕	17	145	145	—
CRSR ⇔	29	157	157	—
F7/F8	136	14Ø	14Ø	—
SPACE	32	16Ø	16Ø	

Figura 24.1

I dettagli del codice ASCII del CBM sono indicati nella Fig. 24.1. Si tratta di un disegno ingrandito della tastiera che mostra i codici generati quando vengono battuti i vari tasti. Ciascun tasto prevede quattro (o tre) numeri. Essi corrispondono a:

(a) Il carattere "non shiftato" (cioè non preceduto da shift)

(b) Il carattere normalmente shiftato (tasto premuto tenendo abbassato SHIFT)

(c) Il carattere "shift Commodore" (tasto premuto tenendo abbassato C=)

(d) Il carattere "Shift Control" (tasto premuto contemporaneamente al tasto CTRL)

Solo alcuni tasti rispondono quando CTRL viene tenuto abbassato. Quelli che non rispondono sono contrassegnati con un ——— al disotto degli altri tre numeri.

Il diagramma mostra che, ad esempio quando viene battuto D con il tasto C= tenuto abbassato, il codice ASCII del CBM del carattere prodotto è 172. La stringa "COMMODORE" verrebbe memorizzata come una sequenza di 9 byte con i valori 67, 79, 77, 77, 79, 68, 79, 82, 69. Il diagramma chiarisce che, i tasti di controllo del cursore e i tasti speciali di funzione alla destra della tastiera producono codici ASCII CBM, quantunque essi non corrispondano ad alcun carattere stampato.
La funzione standard che fornisce il codice ASCII CBM di qualsiasi carattere è ASC, che prende una stringa come argomento e produce il codice ASCII CBM del primo carattere. Così

```
PRINT ASC("X")
```

dà 88 e

```
PRINT ASC("123456")
```

dà 49.

Per ovvii motivi, ASC non può essere applicato ad una stringa nulla (""). Se ci si prova, si ottiene un messaggio

```
?ILLEGAL QUANTITY ERROR.
```

Il programma usato per inserire i numeri nella Figura 24.1 era fondamentalmente questo:

```
10 GET A$:IF A$= "" THEN 10
20 PRINT A$;ASC(A$)
30 GOTO 10
```

Impostare questo programma ed eseguirlo per controllare alcuni dei valori nella Figura 24.1. Occorrerà una certa fantasia per gestire tutti i caratteri di controllo; si potrebbe per esempio rimuovere A$; dalla riga 20.

USO DI ASC - CONTEGGIO DELLA COMPARSA DELLE LETTERE

La funzione ASC è particolarmente utile in due campi: il primo quando si vogliono tradurre singoli caratteri in numeri. Per esempio, è possibile voler violare un codice segreto analizzando un messaggio cifrato e contando il numero delle volte che ogni lettera viene usata. Chiaramente si potrebbe scrivere un programma con numerose istruzioni del tipo.

```
...
IF A$="J" THEN AJ=AJ+1
IF A$="K" THEN AK=AK+1
```

e successivamente

```
...
PRINT "J",AJ
PRINT "K",AK
```

Con la funzione ASC è possibile fare molto meglio. Il diagramma mostra che i codici ASCII CBM per lettere iniziano a 65 per A e procedono fino a 90 per Z. È possibile usare il codice ASCII CBM per ciascuna lettera con un indice per una matrice dove ciascun elemento corrisponde ad una lettere e tenere nota del numero di volte che quella lettera è usata.
Nel programma che segue, ★ è usato come carattere di terminazione. Altri caratteri che non sono lettere, sono visualizzati sullo schermo, ma ignorati. Il programma consente di battere un messaggio e quindi di visualizzare la frequenza di ciascuna lettera:

```
 10 DIM T(26)
 20 GET X$:IF X$=""THEN 20
 30 IF X$="★" THEN 90
 40 PRINT X$;
 50 IF X$ < "A" OR X$ > "Z" THEN 20
 60 P = ASC(X$) - 64
 70 T(P) = T(P) + 1
 80 GOTO 20
 90 PRINT
100 FOR P = 1 TO 26
110 PRINT T(P);
120 NEXT P
130 STOP
```

Glossario

T(26):	Matrice di contatori T(1) per A, T(2) per le B e così via fino a T(26) per le Z.
X$:	Carattere corrente
P:	Codice ASCII del carattere corrente meno 64. È usato come indice per T.

In questa forma il programma produrrà una serie piuttosto disordinata di numeri. È possibile migliorare l'output usando la funzione CHR$ che è l'opposto della funzione ASC: essa converte un numero in codice ASCII nella corrispondente stringa di un carattere. Per esempio, il comando

```
PRINT CHR$(68)
```

dà D.
Possiamo cambiare le ultime righe del programma in modo che risulti

```
100 FOR P=1 TO 26 STEP 2
110 PRINT CHR$(P+64);T(P),
    CHR$(P+65);T(P+1)
120 NEXT P
130 STOP
```


Il programma visualizza ora una tabella ordinata, lunga 13 righe, con due entrate per riga. L'esempio che segue fornisce una visualizzazione tipica:

```
SIBELIUS
WAS VERY REBELIUS
WHEN SCORING FOR THE TIMPANI
IN HIS FIRST SYMPANI

A 3   B 2
C 1   D 0
E 6   F 2
G 1   H 3
I 10  J 0
K 0   L 2
M 2   N 5
O 2   P 2
Q 0   R 5
S 8   T 3
U 2   V 1
W 2   X 0
Y 2   Z 0
```

In questo programma la funzione CHR$ è usata per convertire la sequenza numerica 1,2,3 . . .26 nelle stringhe "A", "B", "C", . . ."Z".

USO DI ASC - PER IGNORARE L'INPUT ILLECITO

La seconda area in cui ASC è utile è nel gestire dati che per qualsivoglia ragione non possono essere trattati dal normale comando INPUT. Per dare un semplice esempio, si pensi a disegnare una interfaccia per gli utenti così ingenui — o così goffi —da non riuscire a scrivere un numero senza battere almeno uno o più tasti sbagliati. Si vuole rendere la cosa più facile facendo in modo che la macchina ignori tutti i tasti salvo le dieci cifre decimali da 0 a 9, il tasto DEL per cancellare gli errori e il tasto RETURN per terminare il numero. Se un tasto viene totalmente ignorato non viene neppure visualizzato sullo schermo, cosicchè l'utente non deve nemmeno rendersi conto di che cosa ha effettivamente battuto.
Una adatta specifica, schema di flusso e subroutine sono indicati qui di seguito.
Notare che tutti i caratteri significativi compresi tra DEL e RETURN, sono rilevati dai rispettivi codici ASCII. Il tasto DEL toglie il carattere più a destra della stringa che viene assemblata. Esso inoltre sostituisce il simbolo visualizzato sullo schermo con uno spazio quindi sposta il cursore all'indietro in modo che il successivo carattere battuto appaia nel posto giusto. Così la cancellazione di un carattere richiede tre caratteri: cursore a sinistra, spazio e cursore a sinistra.

Specifica della subroutine

Scopo: Leggere un numero dalla tastiera, ignorando tutti gli altri caratteri privi di significato

Righe: da 7000 a 7090

Parametri: Output: Risultato fornito in X1

Locali: PP, AA$, XX$

XX$=" "
PP=Ø
Richiama un carattere della tastiera in AA$
AA$= cifra decimale?
Sì
Visualizzazione AA$
XX$= XX$+ AA$
PP=PP+1
No
ASC (AA$)= 2Ø (DEL) ?
Sì
PP=Ø
Sì
No
No
Cancella ultimo carattere su schermo e sposta indietro il cursore. PP=PP−1
Rimuove il carattere più a destra da XX$
ASC (AA$)=13 (RETURN)
No
Sì
PP=Ø?
Sì
No
X1=VAL(XX$)
RETURN
Glossario
XX$: Stringa usata per accogliere cifre decimali valide
PP: Conta il numero di cifre in XX$
AA$: Carattere corrente dalla tastiera

```
7000 REM INPUT NUMERI ROBUSTO
7010 XX$="": PP=0
7020 GET AA$:IF AA$="" THEN 7020
7030 IF AA$>="0" AND AA$<="9"
     THEN PRINT AA$;:XX$=XX$+AA$:
     PP = PP + 1 : GOTO 7020
7040 IF ASC(AA$)<>20 THEN 7070 : REM
     CONTROLLA DEL (=20)
7050 IF PP = 0 THEN 7020 : REM NON
     POSSO CANCELLARE NULLA
7060 PRINT " [SHIFT] e [CRSR ⇐⇒] spazio
     [SHIFT] e [CRSR ⇐⇒]";: PP = PP-1 :
     XX$ = LEFT$(XX$,PP) : GOTO 7020
7070 IF ASC(AA$)<>13 THEN 7020 : REM
     CERCA RETURN
7080 IF PP = 0 THEN 7020 : REM DEVE
     ESSERE QUALCHE CIFRA
7090 X1 = VAL(XX$) : RETURN
```


La relazione tra ASCII CBM e il codice dello schermo è irregolare. Le cinque cifre binarie più a destra sono sempre le stesse, ma le altre cifre non seguono qualsiasi semplice profilo. La situazione è espressa nella tabella che segue:

3 bit super. codice ASCII CBM	Range numer. codice ASCII CBM	3 bit super. codice dello schermo	Commenti
000	0-31	—	Caratteri di controllo
001	32-63	x01	
010	64-95	x00	
011	96-127	—	Non usati caratteri di controllo
100	128-159	—	
101	160-191	x11	
110	192-223	x10	
111	224-255	—	Non usati

Nel codice dello schermo, x=0 per un carattere normale e x=1 per un carattere in negativo.
I seguenti comandi possono essere usati per passare dal codice ASCII CBM (AA) al codice dello schermo (SS):

a) Da codice ASCII CBM a codice schermo:

```
SS=(AA AND 31) + 0.5*(AA AND 128):
   IF (AA AND 64)=0 THEN SS=SS+32
```

b) Da codice schermo a codice ASCII CBM:

```
AA=(SS AND 31) + 2*(SS AND 64) -
   (SS AND 32) + 64
```

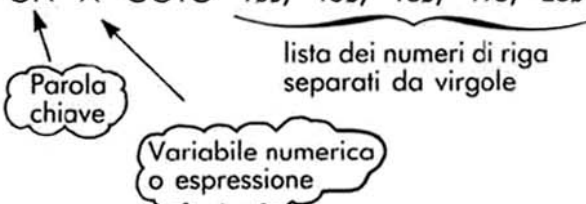
Infine, notare che il codice ASCII del CBM differisce per qualche importante dettaglio dall'ASCII standard quale viene usato su molte altre macchine. Se si collega il COMMODORE **64** ad un altro computer, occorre osservare questo punto attentamente!

ESPERIMENTO 24·1

Scrivere un programma che consenta all'utente di eseguire semplici disegni con una linea piuttosto spessa. Inizialmente il programma visualizza uno spazio in negativo al centro dello schermo. Questo è l'inizio di una linea continua che viene estesa di uno spazio vero l'alto, quando l'utente batte il tasto di funzione F1. Analogamente la linea è estesa a destra, verso il basso o a sinistra rispettivamente in risposta ai tasti F3, F5 e F7. Usare il programma per disegnare una spirale.

Esperimento 24.1 completato	

IL COMANDO "ON"

Un'altra funzione che talvolta è utile è il comando ON. Questo comando consente al programma di saltare in una qualsiasi di parecchie direzioni in relazione al valore di una variabile o espressione.

Il **64** prende il valore della variabile (o espressione) e lo usa per scegliere uno dei numeri di label nella lista. Se il valore è 1, prende il primo, se è 2 prende il secondo e così via. Se il valore è minore di 1 o più alto del numero di label nella lista, non c'è alcun salto.
L'esempio suddetto è equivalente a:

```
IF A = 1 THEN 1ØØ
IF A = 2 THEN 15Ø
IF A = 3 THEN 18Ø
IF A = 4 THEN 195
IF A = 5 THEN 23Ø
```

Se A è minore di 1 o maggiore di 6, verrà eseguita la riga che segue ON.
Nel BASIC del **64** c'è anche una versione del comando ON che usa GOSUB invece di GOTO.
Un uso del comando ON potrebbe essere in un programma che presenta all'utente un "menù" di opzioni come questo:

```
 1Ø PRINT"DESIDERI CONSIGLI PER"
 2Ø PRINT"MEMORIZZARE PROGRAMMI(1)"
 3Ø PRINT"USARE RND(2)"
 4Ø PRINT"DISEGNARE IMMAGINI(3)"
 5Ø PRINT"IL CODICE ASCII(4)"
 6Ø PRINT"PRODURRE SUONI(5)"
 7Ø INPUT"BATTI 1–5"; X
 8Ø ON X GOSUB 3ØØ, 4ØØ, 5ØØ, 6ØØ, 7ØØ
 9Ø GOTO 1Ø
...
3ØØ REM FORNISCE CONSIGLI PER
    MEMORIZZARE PROGRAMMI
...
39Ø RETURN
4ØØ REM FORNISCE CONSIGLI PER USARE RND
...
49Ø RETURN
...
...
7ØØ FORNISCE CONSIGLI PER PRODURRE
    SUONI
...
79Ø RETURN
```

IL COMANDO "END"

La maggior parte dei programmi in questo manuale hanno usato STOP per riportare il controllo alla tastiera al termine di un programma. Un comando alternativo è

END

La differenza è che quando viene eseguito, END non dice in quale riga l'interruzione si è verificata; esso segnala soltanto "READY". È possibile usare STOP o END a piacere.

IL COMANDO "DEF"

La successiva funzione da descrivere, denominata DEF, è francamente una delle parti meno utili del BASIC **64**. Si suggerisce che, a meno che non si abbia una profonda conoscenza matematica e si sia particolarmente interessati alle formule, si salti direttamente alla successiva sezione che riguarda l'uso delle cassette di nastro.
La parola chiave DEF consente di denominare una formula e quindi di farvi riferimento per nome invece che scriverla completamente ogni volta. La definizione è scritta in termini di una variabile "fittizia" che è sostituita da un valore effettivo ogniqualvolta la formula viene usata.
Il nome della formula deve contenere le lettere FN seguite da una o due lettere o da una lettera e da una cifra.

FNA o FNX o FNQC o FNG1

sono tutti nomi di formule appropriati.
Una definizione di formula potrebbe essere la seguente:

Una volta che una funzione è stata inserita in un programma, può essere usata scrivendone il nome con un adatto argomento. Pertanto:

```
2Ø Q = FNB(77)
```

servirà invece di

```
2Ø Q = 1 + 3.73 ★ 77 ↑ 2 + 93/77
```

o

```
3Ø PRINT FNB(S)
```

può essere usato per

```
3Ø PRINT 1 + 3.73 ★ S ↑ 2 + 93 / S
```

o di nuovo

```
4Ø ZZ = FNB(P–Q)
```

è ora un modo valido per scrivere

```
4Ø ZZ = 1 + 3.73 ★ (P–Q) ↑ 2 + 93 / (P–Q)
```

Notare che in ogni caso la variabile fittizia X è sostituita dall'argomento di FNB.
DEF soffre di parecchie limitazioni che ne riducono l'utilità. Tre delle più importanti sono:

a) Non è possibile avere più di una variabile fittizia in una definizione, ad esempio una formula comprendente SQR(X↑2+Y↑2) non è ammessa come parte di una definizione di funzione.

b) Non è possibile definire formule stringa vale a dire non può esistere FNB$ nel BASIC del **64**.

c) Non è possibile avere una subroutine (contrapposta ad un'espressione) per elaborare il valore. Si è limitati ad una formula, quantunque si possa pensare che le subroutine possano andare bene.

ESPERIMENTO 24•2

(Solo per i matematici!)

a) Definire una funzione FNA per elaborare la formula

X↑3 + (X+7)↑2 − 100

Usarla per tabulare il valore di $x^3(x+7)^2-100$ per valori di x compresi fra 2 e 3, procedendo ad intervalli di 0.1. Stimare anche il valore di x per il quale.

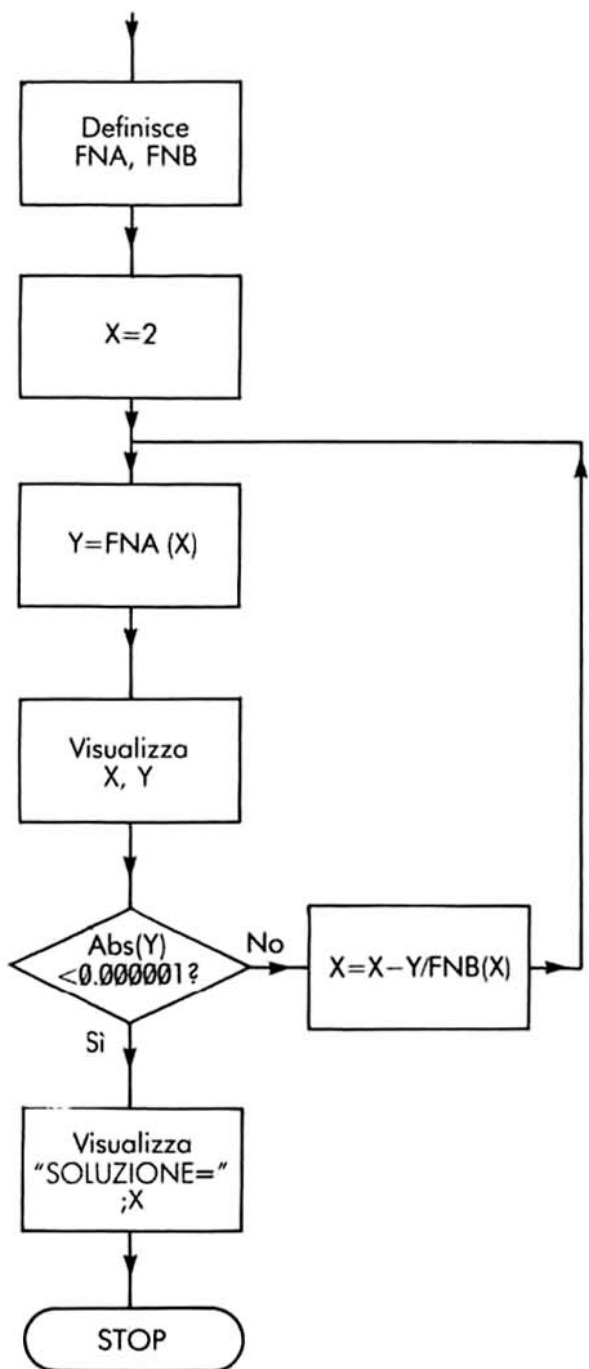
$x^3 + (x-7)^2 - 100 = 0$

b) Definire una seconda funzione FNB per la formula

3 ★ X↑2 + 2 ★ (X + 7)

(Il calcolo denota che si tratta della derivata della prima funzione rispetto a X).
Scrivere ora un programma per calcolare una soluzione approssimata all'equazione

$x^3 + (x - 7)^2 - 100 = 0$

usando il metodo di Newton-Raphson. Uno schema di flusso appropriato è:

Glossario

X: Valore corrente di x
Y: Valore corrente di $x^3 + (x-7)^2 - 100$

Esperimento 24.2 completato	

MEMORIZZAZIONE E RICHIAMO DI DATI SULLA CASSETTA

I lettori non esperti di matematica ritornino qui! Ora osserveremo i comandi per memorizzare e richiamare i dati (invece che i programmi) sulle cassette di nastro. Per esempio, si potrebbe avere una grande raccolta di osservazioni scientifiche o risposte ad un questionario che si vuole analizzare con parecchi programmi diversi. Chiaramente vale la pena di tenere questi dati in forma leggibile dalla macchina in modo da non doverli ribattere ripetutamente.

I principi base per l'uso delle cassette e dei dischetti sono abbastanza simili ma i dettagli pratici sono così diversi che si potrebbe creare confusione esaminandoli contemporaneamente. Per questo è stata dedicata una sezione alle cassette ed un'altra a parte ai dischetti. Occorre studiare la sezione che si applica al proprio computer e trascurare per il momento l'altra.

DATI SU CASSETTA

L'unità base di memoria su una cassetta di nastro è il "file". Esso è costituito da tre sezioni:

TESTATA	CORPO DEL FILE	CODA
nome del file	caratteri	marcatore di fine file

Direzione del movimento del nastro →

Questo diagramma mostra un segmento di nastro "svolto" dalla cassetta. La testata viene per prima e identifica il file contenendone il nome. Il nome può essere qualsiasi stringa di caratteri di una ragionevole lunghezza, ad esempio "DATI SATELLITE" o "INDIRIZZI CLUB TENNIS".

Successivamente viene il CORPO DEL FILE. Esso contiene una sequenza di caratteri e può essere lungo a piacere fino ad un'intera cassetta (90 minuti). Ciascun minuto di registrazione contiene circa 1200 caratteri.

Il corpo del file è diviso in "blocchi" di dimensioni uguali ciascuno dei quali contiene 256 caratteri. Ciascun blocco è seguito da uno spazio che consente al meccanismo del nastro di fermare e far ripartire il nastro fra i blocchi. Infine la CODA contiene un gruppo speciale di simboli che contrassegnano la fine del file. In pratica non occorre sapere molto a proposito dei dettagli in quanto il sistema funziona automaticamente.

Una singola cassetta di nastro può contenere parecchi file diversi registrati uno dopo l'altro. Il solo possibile inconveniente con questa configurazione è che per leggere i file che si trovano in fondo alla coda occorre far passare per prima cosa quelli che si trovano all'inizio.

PRINT # - PER SCRIVERE I DATI

Per scrivere un file su una cassetta, si usano tre nuovi comandi:

```
OPEN 1,1,1"nome file"
PRINT # 1,
CLOSE1
```

Per iniziare un file, il programma deve dare un comando OPEN nella forma sopra indicata. Il nome del file può essere scelto liberamente ma i numeri 1,1 e 1 devono essere scritti precisamente come indicato.

Quando viene eseguito OPEN, il **64** fa comparire un messaggio sullo schermo che dice

PRESS RECORD AND PLAY ON TAPE

Caricare una cassetta vergine nel registratore (o se non è vergine una che è stata riavvolta oltre i file o i programmi che verranno ancora usati) e premere i tasti di controllo corretti. Il nastro deve essere sufficientemente lungo in relazione al file che si vuole scrivere, dato che non c'è modo di cambiare i nastri a metà del file. Ricordarsi di premere RECORD in modo che rimanga abbassato. Se ci si dimentica di farlo, il **64** esegue tutti i movimenti per scrivere un file ma non inserisce effettivamente alcuna informazione sul nastro. Fare attenzione!
È possibile iniziare a trasmettere i dati con il comando

```
PRINT#1,
```


Notare che tutti gli 8 caratteri in questa parola chiave sono inseparabili e devono essere battuti esattamente come indicato. In particolare la virgola è essenziale e non è possibile usare il punto interrogativo invece di PRINT.
La parola chiave deve essere seguita dai nomi delle variabili che si vogliono scrivere. Se ce n'è più di una nel comando, i nomi devono essere separati dalla sequenza; ",";. Le variabili possono essere numeri, stringhe od una combinazione di entrambi. Ecco alcuni esempi:

```
PRINT # 1,X
PRINT # 1,P$
PRINT # 1,Q$(J);",";X(J);",";R$(J+1)
```

È possibile avere un numero di variabili a piacere nel comando PRINT # 1, posto che

a) La lunghezza del comando non superi gli 8Ø caratteri (questo è il limite normale che si applica a tutti i comandi)

b) Il numero totale di caratteri inviato al nastro da un qualsiasi comando sia inferiore a 8Ø.

Se si viola la seconda regola nulla sembra andare storto quando si scrivono i dati ma non è più possibile leggerli successivamente. Far quindi attenzione! È possibile, naturalmente, usare PRINT#1 ripetutamente all'interno di una iterazione per scrivere tutte le informazioni desiderate. Se si scrivono più di poche variabili si noterà che il nastro si muove a scatti e ciò in quanto il **64** ha un serbatoio interno di informazioni che agisce come "buffer" ossia come una memoria di transito tra il programma e il nastro. Quando la macchina esegue i comandi PRINT#1, i dati usati vengono per prima cosa raccolti nel buffer. Quando il buffer è pieno i suoi contenuti vengono inviati alla cassetta in una singola "scarica" per scrivere un blocco. Quindi il nastro si ferma, il buffer viene svuotato e il processo riparte da capo. Quando sono state scritte tutte le informazioni che si vogliono registrare, impartire un comando CLOSE1. Ciò forza il **64** a scrivere un altro blocco (quantunque il buffer possa essere soltanto parzialmente pieno) e una coda con marcatore di fine file.

INPUT # PER LEGGERE I DATI

Per richiamare le informazioni dalla cassetta di nastro, occorrono le tre istruzioni:

```
OPEN 1,1,Ø, "NOME FILE"
INPUT # 1,
CLOSE1
```

Il comando OPEN con lo Ø davanti al nome di file (invece di 1) fa sì che il **64** apra un file per la *sola lettura*. La macchina visualizza il messaggio

```
PRESS PLAY ON TAPE
```

ed attende che s'inserisca la cassetta nel registratore e si prema PLAY. *Non premere* RECORD se non si vogliono perdere dati preziosi. Quando il **64** rileva che il nastro è caricato, inizia a cercare sul nastro un file con un nome che corrisponde a quello nel comando OPEN. Il processo di abbinamento richiede soltanto che la stringa nel comando OPEN sia la stessa dell'*inizio* del nome del file. Se il nome effettivo del file è "DATI GIOVEDI" il file sarà aperto da uno qualsiasi dei seguenti comandi:

```
  OPEN 1,1,Ø,"DATI GIOVEDI"
o OPEN 1,1,Ø,"DATI"
o OPEN 1,1,Ø,"D"
o OPEN 1,1,Ø,"":REM LA STRINGA NULLA
      APRE OGNI FILE
o OPEN 1,1,Ø:REM PUOI OMETTERE TITOLO
o X$= "DAT": OPEN 1,1,Ø, X$ : REM
      PUOI USARE UNA VARIABILE
```

Se il titolo è indicato come stringa nulla o viene omesso, il comando aprirà il primo file che incontra, qualunque sia il suo nome.
Il comando INPUT # 1, è come il comando PRINT # 1 ma alla rovescia. La parola chiave è seguita dai nomi delle variabili da leggere dal nastro, separati da virgole, se ce n'è più d'uno. Ne sono esempi:

```
INPUT # 1,Z
INPUT # 1,P$
INPUT # 1,R$,Q,T$
```

Notare che il numero e il tipo delle variabili che seguono INPUT # 1, deve essere indentico a quello usato per inserire i valori sul nastro all'inizio. Il comando

```
INPUT # 1,A$,B$,C:REM DUE STRINGHE
E UN NUMERO
```

potrebbe essere usato per estrarre i dati originariamente scritti da

```
  PRINT # 1,A$;",";B$;",";C :REM DUE
      STRINGHE E UN NUMERO
o PRINT # 1,Z$(Q);",";"PORTA A";",";
      X:REM DUE STRINGHE E UN
      NUMERO
```

ma se i dati fossero stati scritti nella forma Y;",";P$ il suddetto INPUT non funzionerebbe. Quando il sistema legge da una cassetta di nastro, possono succedere varie cose impreviste. Per tener conto di questa difficoltà, il **64** riserva una variabile speciale denominata ST (per "Status=Stato") e la usa per dare un rapporto codificato ogni volta che viene eseguito il comando INPUT=1. Il valore 0 significa che tutto va bene. 64 segnala che si è raggiunto la fine del file e altri valori implicano che qualcosa è andato male: il nastro si è corrotto, o probabilmente non era stato correttamente registrato all'inizio. Per illustrare l'azione del registratore a cassetta, ecco un paio di programmi. Il primo consente di disegnare una figura sullo schermo usando il cursore e i comandi del colore e quindi registrare questa immagine su un file "prendendo" (PEEK) i valori dello schermo e le RAM dei colori e scrivendoli come numeri. Il secondo programma legge il file e ricostruisce l'immagine. Studiare entrambi i programmi attentamente e notare il modo in cui ST è stato usato. Quindi, impostarli uno per uno e provarli.

```
10 OPEN 1,1,2,"IMMAGINE SCHERMO"
```

20 PRINT" e CRSR
DISEGNA IMMAGINE A PIACERE"

```
30 PRINT"USANDO I COMANDI DEL
   CURSORE"
40 PRINT"E DEL COLORE."
50 PRINT"LASCIA CURSORE SU RIGA"
60 PRINT"SUPERIORE CHE
70 PRINT"DEVE ESSERE VUOTA."
80 PRINT"QUINDI PREMI RETURN"
85 FOR S=1 TO 5000: NEXT S
```

90 INPUT" ";X$:
REM UTENTE DISEGNA IMMAGINE

```
100 FOR J=0TO 999:REM ESAMINA RAM
    SCHERMO E COLORE
110 PRINT # 1,PEEK(1024+J);",";PEEK
    55296+J)
120 NEXT J
130 CLOSE1
140 STOP
```

ora riavvolgere il nastro e battere il seguente programma:

```
10 OPEN1,1,0,"IMMAGINE SCHERMO"
20 J=0
30 INPUT # 1,X,Y:REM RICAVA CODICI
   COLORE E SCHERMO
40 POKE 1024+J,X:POKE 55296+J,Y
50 IF ST<>0 THEN 70
60 J=J+1:GOTO 30
70 IF ST=64 THEN 100:REM CONTROLLA
   FINE FILE
80 PRINT"DIFETTO NASTRO"
90 STOP
100 CLOSE1
110 GOTO 110:REM ARRESTA ITERAZIONE
    SE TUTTO OK
```

GET #

Un altro comando che talvolta è usato con le cassette a nastro è

GET # 1,

Questo comando è piuttosto simile a INPUT # 1, salvo che trasferisce singoli caratteri. Esso apparirebbe in sequenze tipo

```
...
100 GET A$:IF A$=""THEN 100:REM
    OTTIENI UN CAR. DA TASTIERA
110 PRINT A$:REM VISUALIZZALO
120 PRINT #:1,A$:=REM INVIALO A
    CASSETTA DI NASTRO
...
```

e

```
...
200 GET # 1,X$:REM OTTIENI UN
    CARATTERE DA NASTRO
210 IF ST<>0 THEN 300:REM SALTA SE
    FINE FILE O ERRORE
220 PRINT X$;
...
```

PROBLEMI

Come si è visto, il **64** fornisce le funzioni base per scrivere dati su una cassetta e per richiamarli successivamente. Queste funzioni sono primitive e hanno taluni inconvenienti:

1) Lettura e scrittura sono lente:

2) L'affidabilità del sistema a cassetta non è perfetta. Le cassette possono essere già danneggiate al momento dell'acquisto oppure possono essere danneggiate dall'umidità, da una manipolazione scorretta, dell'eccessivo caldo o freddo o da forti campi magnetici. Tutte queste circostanze possono produrre errori nei file. Il tasso di errore nel memorizzare i dati è molto più alto di quello che si ha nei programmi, in quanto

   a) I file di dati sono generalmente più lunghi.

   b) Non c'è modo di verificare i file di dati come è possibile fare con i programmi

3) Il **64** può gestire soltanto una cassetta. Ciò significa che non è possibile correggere un file o aggiungervi dati a meno che non sia sufficientemente corto da poterlo inserire completamente nella memoria del **64.**

Chi è seriamente interessato a memorizzare grandi quantità di dati deve comprare un'unità a floppy disk. La Commodore dispone di una buonissima unità del genere specificamente studiata per il **64**. Se si decidesse di procedere con le cassette, usare nastro della migliore qualità che è possibile trovare, tenere il registratore a cassetta pulito e in perfette condizioni e soprattutto prepararsi spiritualmente a qualsiasi inconveniente occasionale.

DATI SU DISCHETTI

Se si dispone di un'unità a dischetti per il COMMODORE **64**, si avrà familiarità con gli aspetti più importanti dell'uso dei dischetti. Si saprà certamente come averne cura, come formattarli e inizializzarli e come caricare e memorizzare programmi.

In questa sezione si discuteranno i vari modi per usare i dischetti per memorizzare i dati anziché i programmi. L'unità a dischetti è per sua natura un dispositivo complesso ed occorre iniziare inserendovi alcuni importanti informazioni di base.

Come già si sa, ogni comando disco comprende alcuni numeri misteriosi. Per esempio, occorre sempre battere

```
LOAD"MYPROGRAM"8, oppure
OPEN 1,8,15,"I"
```

È ora il momento di spiegare cosa significano questi numeri.

Ogniqualvolta una periferica viene collegata al computer, si devono comprendere tre concetti fondamentali.

Innanzitutto c'è il concetto di un numero di canale (detto talvolta erroneamente "numero file logico" dato che non sempre si riferisce ad un file e non è più logico di qualsiasi altro tipo di numero). Un computer potrebbe essere paragonato ad un agente di viaggio che siede dietro la sua scrivania. Può ricevere informazioni dalle fonti più diverse: dal cliente che gli sta di fronte, da due o tre telefoni e da un terminale collegato a qualche computer remoto. Può anche trasmettere messaggi in modi diversi: per esempio parlando al cliente o battendoli nella console del computer.

Nell'analogia col **64**, ciascuna di queste fonti o destinazioni di informazione viene chiamata canale. Ad esempio il cliente potrebbe essere il canale 1, il terminale di computer il canale 2 e i telefoni i canali da 3 a 5.

Quando un programma deve comunicare con una periferica, deve per prima cosa predisporre un appropriato canale. Un programma BASIC può usare un grande numero di canali verso le varie periferiche (un massimo di 10), che possono essere numerati in qualsiasi modo, posto che i numeri siano tutti diversi e risiedano nel campo da 1 a 127.

Secondo, c'è il concetto di numero di periferica o di dispositivo. Ogni unità collegata al **64** ha un proprio numero di periferica fisso, incorporato in fabbrica e che non può essere facilmente cambiato. Per esempio, i numeri di periferica della cassetta di nastro, della stampante e dell'unità dischi sono sempre rispettivamente 1,4 e 8.

A questo punto ci si potrebbe chiedere perché non uguagliare il numero di canale al numero di periferica e semplificare il tutto?

Il motivo è che tale configurazione non fornisce sufficiente flessibilità. Come si vedrà, spesso ci sono più canali che periferiche!

Il terzo concetto importante è il cosidetto "indirizzo secondario". In alcuni tipi di periferiche, può essere usato un canale di informazioni in modi diversi; per esempio, un registratore a cassetta può leggere o scrivere ma non può fare entrambe le cose contemporaneamente. Lo "indirizzo secondario" di un canale indica esattamente come deve essere usato quel canale. Notare che il significato di indirizzo secondario uguale 0 o 1 non è fisso ma dipende dal dispositivo effettivo cui si riferisce.

Quando viene eseguito un programma BASIC, si può immaginare ciascun canale come un filo telefonico che arriva al computer principale o che ne esce. La Figura 24.2 mostra il profilo generale.

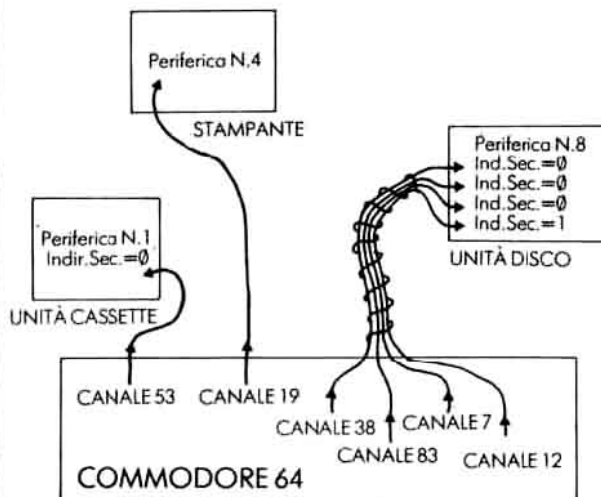



Figura 24.2

Si vedrà facilmente che ciascun filo realmente ha due o tre etichette differenti. Nel computer il filo è contrassegnato con il proprio numero di canale. Nel momento in cui esce viene codificato dal numero di periferica al quale è diretto. Quando arriva alla periferica, può essere contrassegnato una terza volta con un "indirizzo secondario". Naturalmente non occorre cambiare il diagramma del modo effettivo in cui le periferiche sono fisicamente collegate al **64**.

In pratica, tutti i canali (salvo quello che collega il registratore a cassetta) condividono la stessa coppia di fili, che collega tutte le periferiche in una configurazione a festone. Questi fili sono detti "bus seriale".

IL COMANDO "OPEN"

È ora possibile spiegare la struttura e il significato di comandi tipo OPEN. Il comando OPEN imposta un nuovo canale verso un dispositivo o periferica esterna. La parola chiave OPEN è sempre seguita da due o tre numeri e talvolta da una stringa. Il primo numero è quello del canale, il secondo è il numero della periferica fisica e il terzo è l'indirizzo secondario. La stringa, posto che ce ne sia una, è un messaggio che il computer trasmette al canale quando viene aperto.
In alcuni casi, quando deve essere aperto un canale su una periferica che non usa indirizzi secondari, il comando OPEN ha bisogno soltanto di due numeri. Per esempio, per usare la stampante (periferica n.4) come numero di canale 25 occorrerebbe scrivere

```
OPEN 25,4
```

Per selezionare un particolare indirizzo secondario nell'ambito di una periferica, occorrono tutti e tre i numeri dopo la parola chiave. Per es.

```
OPEN 33,8,5
```

aprirebbe il canale 33 come indirizzo secondario 5 dell'unità di controllo disco.
Ora si esaminerà in dettaglio l'unità di controllo disco.
L'unità disco comprende 7 indirizzi secondari numerati da Ø a 5 e 15. Tre di essi sono usati per scopi speciali:
L'indirizzo secondario Ø è usato per il CARICAMENTO di programmi.
L'indirizzo secondario 1 è usato per il SALVATAGGIO (SAVE) di programmi.
L'indirizzo secondario 15 è usato come canale di controllo, che gestisce tutte le funzioni speciali tipo caricamento dell'elenco, inizializzazione di dischi, segnalazione di errori che si verificano su altri canali. Ogniqualvolta il dischetto è usato per qualsiasi scopo (salvo che per il caricamento o salvataggio di un programma), deve essere impostato un canale collegato all'indirizzo secondario 15 con un comando del tipo

```
OPEN 1,8,15
```

oppure

```
OPEN 1,8,15,"IØ"
```

che specificano entrambi il canale 1.
Il secondo inizializza il dischetto oltre ad aprire il canale.
Gli altri 4 indirizzi secondari (da 2 a 5) possono essere usati per trasmettere dati a/dal dischetto. Per far ciò, il programma deve avere almeno 2 canali contemporaneamente aperti verso l'unità a disco - il canale di controllo e un canale di dati. Ciò è abbastanza normale; in effetti è possibile aprire un canale separato per ciascun indirizzo secondario. È possibile avere aperti contemporaneamente fino a 3 canali dati più il canale di comando.
L'unità base di memoria su disco è il file, di cui esistono due tipi principali: file di programma (che possono essere salvati (SAVE) e caricati (LOAD)) e file di dati sequenziali che memorizzano elementi tipo numeri e stringhe. Ogni file ha un nome che può avere una lunghezza massima di 16 caratteri e viene conservato in un elenco su una parte speciale del dischetto. Un file può essere lungo a piacere posto che tutti i file sul dischetto riuniti, non superino circa 16Ø.ØØØ caratteri.

PRINT# PER SCRIVERE I DATI

Quando il programma deve usare il dischetto per memorizzare i dati, occorrerà creare un nuovo file:
Innanzitutto, aprire un canale verso l'indirizzo secondario 15, che gestisce tutti i problemi organizzativi. È inoltre possibile inizializzare contemporaneamente il dischetto.
Successivamente, aprire un altro canale verso uno degli indirizzi secondari normali (ad esempio il numero 2). La stringa che segue l'indirizzo secondario deve fornire i dettagli del file che si desidera scrivere. Occorre sempre indicare un nome di file, un tipo (che è sempre SEQ) e un modo che è sempre WRITE se si sta creando un nuovo file. Questi tre elementi sono separati da virgole e poste all'interno di virgolette.
Ad esempio, si voglia creare un file di dati denominato DINOSAUR. Il programma relativo inizierebbe con:

```
1Ø OPEN 1,8,15,"IØ":REM APRI
   CANALE CONTROLLO E INIZIALIZZA
2Ø OPEN 2,8,3,"DINOSAUR,SEQ,
   WRITE":
   REM CREA NUOVO FILE
```

Notare che il numero del canale di controllo (1) potrebbe assumere qualsiasi valore compreso tra 1 e 127; il numero del canale di dati (2) qualsiasi valore nello stesso campo salvo quando allocato al canale di cotrollo e l'indirizzo secondario (3) qualsiasi valore compreso tra 2 e 5. Per contro, non c'è scelta per quanto riguarda il numero di periferica (sempre 8 per l'unità disco) e per il canale di controllo (sempre 15).
Una volta aperto un file di dati è possibile usare il comando PRINT# per trasmettere informazioni da registrare. PRINT# deve essere seguito dal numero di canale del file di dati e quindi dalle variabili o espressioni che si desiderano registrare. Il numero di canale deve essere sempre seguito da una virgola e i nomi variabili (se ce n'è più di uno) dalla sequenza; ",";. Ecco alcuni esempi

```
PRINT#2,X
PRINT#2,P$;",";X+5
PRINT#2,Q$;",";S$",";J;",";K
```

Notare che PRINT# non corrisponde assolutamente a PRINT. In particolare se non si devono scrivere cose del tipo "?#" al posto di PRINT#; la cosa non funziona. Non ci devono essere inoltre spazi tra PRINT E # o tra # e il numero di canale.
È possibile avere variabili a piacere nel comando PRINT# posto che la lunghezza totale del comando non superi gli 80 caratteri (il limite normale per i comandi).
Quando il programma esegue un comando PRINT#, trasmette i valori delle variabili citate al file specificato dal numero del canale. Nella maggior parte dei programmi il comando PRINT# verrà inserito all'interno di un'iterazione cosicché verrà richiamato ripetutamente. In questo modo, è possibile creare un file di qualsiasi lunghezza.
Quando un programma ha terminato di creare un nuovo file, deve impartire un comando CLOSE. Ciò assume la forma

```
CLOSE n
```

dove n è il numero di canale del nuovo file. Ne sono alcuni esempi

```
CLOSE 5
```

o

```
CLOSE 6
```

Quando il programma ha terminato tutte le comunicazioni con l'unità di controllo disco, deve chiudere il canale verso l'indirizzo secondario 15, usando un comando CLOSE.

INPUT# PER LEGGERE I DATI

Per richiamare informazioni da un dischetto, il programma deve iniziare aprendo un canale di cotrollo verso l'unità di controllo disco esattamente come sopra descritto. Quindi deve aprire un canale dati verso il file che si desidera leggere. Il nome file deve essere identico a quello che è stato creato precedentemente e il modo deve essere READ. Le appropriate istruzioni potrebbero essere

```
OPEN 2,8,15,"I0"REM APRI CANALE
      CONTROLLO
OPEN 5,8,11,"DINOSAUR,SEQ,READ":
      REM APRI DINOSAUR PER LETTURA
```

quale canale o indirizzo secondario sono stati scritti.
Il comando INPUT# è l'inverso di PRINT#. La parola chiave è seguita dai numeri delle variabili da leggere dal dischetto, separati da virgole se ce n'è più di una. Alcuni esempi sono:

```
INPUT#1,Z
INPUT#1,P$
INPUT#1,R$,Q,T$
```

Notare che il numero e il tipo delle variabili che seguono la parola chiave devono essere identici a quelli usati inizialmente per inserire i valori sul dischetto. Per esempio, il comando

```
INPUT#4,A$,B$:REM ESTRAI DUE
STRINGHE E UN NUMERO
```

potrebbe essere usato per richiamare i dati originariamente scritti da

```
PRINT#2,A$;",";B$;C:REM
REGISTRA DUE STRINGHE E UN
NUMERO
```

o

```
PRINT#99,Z$(Q);",";"DA";",";
55:REM DUE STRINGHE E UN NUMERO
```

ma non

```
PRINT#22,A;",";b$:REM UN NUMERO E
UNA STRINGA
```

A questo punto vale la pena di fornire un semplice esempio. Battere il seguente programma, che consente di conservare su un file un'immagine completa dello schermo:

```
10 OPEN 1,8,15,"I0":
     REM OCCHIO ZERO
20 OPEN 2,8,3,"SCREEN,SEQ,WRITE"
30 PRINT" SHIFT e CLR HOME , CRSR "
40 PRINT"DISEGNA IMMAGINE
     A SCELTA"
50 PRINT"USANDO COMANDI
     CURSORE E"
60 PRINT"COLORE"
70 PRINT"LASCIA CURSORE
     SU RIGA"
80 PRINT"SUPERIORE
     CHE DEVE ESSERE"
90 PRINT"VUOTA.QUINDI
     PREMI RETURN"
100 FOR S=1 TO 5000:NEXT S
110 INPUT" SHIFT e CLR HOME ";X$
120 FOR J=0TO 999:REM ESAMINA
130 PRINT#2,PEEK(J+1024);",";PEEK
     (J+55296):REM SCRIVI DETTAGLI
     DI UN CARATTERE
140 NEXT J
150 CLOSE2:REM CHIUDI CANALE DATI
160 CLOSE1:REM IDEM CANALE
     CONTROLLO
170 STOP
```

Caricare ora un dischetto inizializzato ed eseguire il programma. Quando richiede di disegnare un'immagine, farlo con il comando del cursore, i tasti dei colori e qualsiasi simbolo grafico desiderato. Tenere libera la riga superiore dello schermo.
Quando pronti, spostare il cursore alla riga superiore e premere RETURN. Il programma troverà la sua strada sullo schermo, ricavando il codice del carattere e il colore di ciascun carattere e scrivendoli in un file detto SCREEN.

Ciò richiederà un minuto o due, durante il quale si vedrà illuminarsi la spia dell'unità disco.
Quando il programma termina, caricare l'elenco dal dischetto e listarlo. Fra i vari programmi memorizzati si vedrà un nuovo file denominato SCREEN, che conterrà circa 40 blocchi (in relazione ai dettagli dell'immagine) e sarà del tipo SEQ (e non del tipo PRG come gli altri).
Per richiamare l'immagine, battere il seguente programma:

```
10 OPEN 1,8,15,"I0"
20 OPEN 2,8,2,"SCREEN,SEQ,READ"
30 J=0
40 INPUT#2,X,Y
50 POKE 1024+J,X
60 POKE 55296+J,Y:REM RIPRISTINA
   CARATTERE E COLORE
70 IF ST< >0THEN 100
80 J=J+J
90 GOTO 40
100 IF ST=64 THEN 120
110 PRINT"DISK FAULT"
120 CLOSE 2
130 CLOSE 1
140 GOTO 140
```

Durante l'esecuzione, questo programma legge la descrizione dell'immagine dal file e la ripristina sullo schermo. Notare che il file è ora aperto in un modo diverso.
Quando il computer sta leggendo un file, possono succedere varie cose impreviste. Per esempio, se il dischetto è stato danneggiato potrebbe esserci informazioni illeggibili oppure si potrebbe semplicemente arrivare alla fine del file.
Per risolvere questi problemi, il COMMODORE è stato munito della variabile speciale ST (che sta per "Stato"). Ogniqualvolta il computer agisce su un file (ad esempio legge o scrive alcuni valori) viene impostato automaticamente ST per mostrare cosa è successo. Il valore 0 significa che tutto è andato come previsto. Esce 64 quando si legge l'ultimo bit o elemento del file e i guasti "reali" sono indicati dai vari altri codici. Il programma appena provato usa ST per interrompere la lettura non appena lo schermo è pieno.
Questa sezione ha fornito la più breve introduzione possibile all'uso dei file su dischetto. In pratica, è la presenza dell'unità controllo disco che trasforma il COMMODORE **64** in un computer serio in grado di eseguire l'elaborazione testi e numerose applicazioni scientifiche e gestionali. Per saperne di più sul sistema di dischi, leggere il manuale di riferimento VIC 1541.

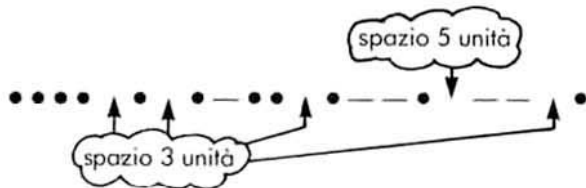
ESPERIMENTO 24·3

In questo esperimento si disegnerà e si costruirà un insegnante meccanico del Codice Morse.
Il Codice Morse è usato per trasmettere informazioni per radio. Ciascuna lettera dell'alfabeto ha un codice che si compone di punti e di linee. Il codice completo è

A	•—	J	•———	S	•••
B	—•••	K	—•—	T	—
C	—•—•	L	•—••	U	••—
D	—••	M	——	V	•••—
E	•	N	—•	W	•——
F	••—•	O	———	X	—••—
G	——•	P	•——•	Y	—•——
H	••••	Q	——•—	Z	——••
I	••	R	•—•		

L'unità base di tempo è il punto e altri intervalli di tempo sono definiti come segue:

Linea	3 punti
Distanza tra punti e linee della stessa lettera	1 punto
Distanza tra lettere	3 punti
Distanza tra parole	5 punti

Per esempio il messaggio HELP ME, verrebbe trasmesso nella forma:

Nel Codice Morse la punteggiatura è ignorata.
Per iniziare, scriveremo una subroutine costituita da due parametri:

a) Una stringa contenente una frase

b) Un numero che dà la velocità desiderata di trasmissione in millesimi di secondo per punto.

La subroutine converte la frase in Morse e la trasmette sul generatore di suoni del **64** alla velocità richiesta.

SUGGERIMENTO: Impostare una subroutine che emetta un suono — o una pausa — di un'adatta lunghezza. Gestirla con programma che usa una tabella bidimensionale come questa:

```
1 3 0 0 0   (A)
3 1 1 1 0   (B)
3 1 3 1 0   (C)
---------
3 3 1 1 0   (Z)
```


La tabella contiene il codice per ciascuna delle lettere da A a Z e deve essere impostata usando le istruzioni READ e DATA.
Una volta soddisfatti della subroutine, passare alla seconda parte dell'esperimento. Questo comporta due programmi:

a) un programma per immettere un testo (un'intera serie di frasi) tramite tastiera e la loro registrazione in un file su cassetta. Usare la frase fittizia "ZZZZ" per terminare l'input.

b) Un programma per leggere le frasi e trasmetterle in Codice Morse a qualsiasi velocità desiderata.

Quando i programmi funzionano, possono essere usati per preregistrare messaggi e trasmetterli ad altissima velocità risparmiando tempo e aumentando la capacità di un canale radio. Essi sono anche utili se si vuole far pratica nel ricevere segnali Morse, dato che è possibile iniziare lentamente e gradualmente accelerare la velocità. Per ottenere i migliori risultati occorre servirsi di qualcuno che batta le frasi da trasmettere in modo da non sapere cosa aspettarsi in anticipo.

Esperimento 24.3 completato	

ESPERIMENTO 24·4

Questo esperimento invita a disegnare e a scrivere un programma di "appuntamenti" col computer.

PER CHI POSSIEDE L'UNITÀ A CASSETTA
Caricare il programma MAKENAMES dalla cassetta di nastro. Inserire una cassetta vergine nel registratore ed eseguire il programma. Essa produrrà una lista di 100 persone e scriverà i rispettivi dettagli personali sulla cassetta in un file denominato "COMPUTER DATA". Il programma viene eseguito in circa 5 minuti e lo schermo risulterà vuoto per la maggior parte del tempo.

PER CHI POSSIEDE L'UNITÀ A DISCHETTI
Caricare il programma MAKENAMES dal dischetto. Quindi inserire un dischetto inizializzato (uno sul quale è possibile scrivere) nell'unità disco ed eseguire il programma. Esso produrrà una lista di 100 persone e scriverà i rispettivi dettagli personali su disco in un file denominato "COMPUTER DATES".
Il record per ciascuna persona si compone delle seguenti voci (nell'ordine in cui sono registrate):

NAME (nome) (stringa)

ADDRESS (indirizzo) (stringa)

TOWN (città) (stringa). Una fra EDINBURGH, GLASGOW, DUNDEE o ABERDEEN

SEX (sesso) (stringa). Una fra F o M (femmina o maschio)

AGE (età) (numero)

HEIGHT (altezza) (numero). Altezza in pollici

MAIN HOBBY (hobby principale (stringa) SECOND HOBBY (hobby secondario (stringa)	Entrambi ricavati dalla seguente lista: FOOTBALL, TENNIS, HILLWALKING (passeggiata in montagna), OPERA (opera), JAZZ (Jazz), ROCK (rock) THEATRE (teatro), READING (lettura), POLITICS (politica),

	STUDYING (studio), CHESS (scacchi), GAMBLING (giochi d'azzardo), HORSE-RACING (corse dei cavalli), CARS (auto), MOTOR-BIKES (moto), CYCLING (bicicletta), MEETING PEOPLE (incontrare persone)
POLITICS (politica) (stringa)	Uno fra CONSERVATIVE (conservatore), LABOUR (laburista), LIBERAL (liberale), SDP, OTHER (altri), NONE (nessuno)

Una volta ottenuto il file delle persone, iniziare scrivendo il programma più semplice che apre il file, legge e visualizza i record uno alla volta. Successivamente disegnare e scrivere un programma di "appuntamenti" che chiede i particolari personali di un "cliente" e quindi cerca il file e preleva la "persona più adatta.
Si supponga che la persona scelta debba vivere nella stessa città. Le persone che soddisfano questi vincoli segnano punti di "bonus" sulla scala arbitraria seguente:

Compatibilità di età: se la ragazza ha la stessa età o non è più giovane di 4 anni dell'uomo:	3 punti
Compatibilità di altezza: se la ragazza ha la stessa altezza o non è più piccola dell'uomo di 4 pollici:	2 punti
Hobbies: per ciascuna hobby condiviso:	5 punti
Politica: per un punto di vista politico comune:	3 punti
se uno vota per i laburisti e l'altro per i conservatori: *meno*	4 punti

L'accoppiamento "migliore" è quello con il punteggio più elevato.

Attenzione: Se si trova qualcuno che realmente piace, è bene rinunciare subito a mettersi in contatto: le persone sul file non sono reali.

Esperimento 24.4 completato	

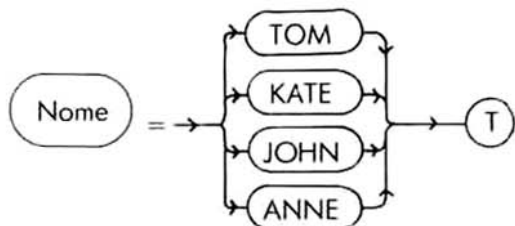
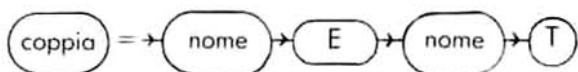
UNITA': 25

	PAGINA
Progetto di programmi - casi pratici	273
Frasi casuali - grammatica linea del tram	273
Probabilità	274
Esperimento 25.1	280
Giochi di avventura o di labirinto	281
Esperimento 25.2	283
Esperimento 25.3	284

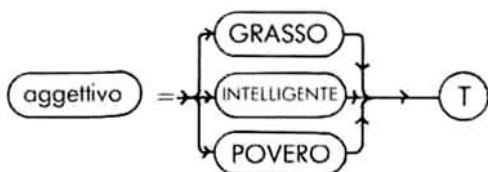
PROGETTO DI PROGRAMMI - CASI PRATICI

L'argomento del progetto di programmi è stato un tema ricorrente nel corso di questo manuale. Questa unità finale comprende alcuni casi pratici ciascuno dei quali mostra come un problema apparentemente complesso può essere risolto rapidamente e facilmente esaminandone la struttura e trasferendo questa struttura al programma stesso

FRASI CASUALI - LA GRAMMATICA DELLA LINEA DEL TRAM

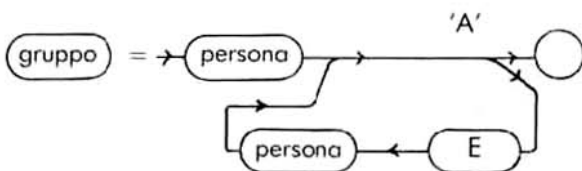
Il primo dei nostri esempi pratici riguarda la produzione di frasi casuali come quelle nell'Unità 6. Chiaramente queste frasi non possono rappresentare semplici raccolte di parole messe insieme in qualsiasi ordine; se devono avere un senso, devono seguire le regole della grammatica.
La notazione spesso usata per queste regole è detta una grammatica "a linea del tram". Si supponga che in un certo punto in una frase che viene costruita occorra scegliere il nome di una persona dalla lista di TOM, KATE, JOHN, ANNE, possiamo scrivere questa parte della grammatica con l'aiuto del diagramma

Si immagini un tram che entri nel diagramma da sinistra (nella direzione della freccia). Quando il guidatore arriva ad uno scambio, la direzione che egli prende, viene decisa a caso. Il tram al limite può arrivare al capolinea alla destra, ma può farlo attraverso uno qualsiasi di quattro percorsi: TOM, KATE, JOHN o ANNE.
In questo diagramma ciascun ovale contiene una parola che è una possibile candidata per parte della frase che viene costruita. Gli ovali nei diagrammi della linea tranviaria possono anche contenere i nomi di altri diagrammi. La differenza è sempre chiara in quanto i nomi dei diagrammi sono scritti in minuscolo.
Osservare il seguente diagramma a linee tranviarie:

(dove "nome" è il diagramma con TOM, KATE, JOHN e ANNE).
Per il guidatore del tram, il primo nome è un tipo di subroutine. Nel momento in cui il tram ha trovato la sua strada attraverso il diagramma delle coppie, può essere arrivato con una delle seguenti frasi:

TOM E JOHN
ANNE E TOM
o addirittura KATE E KATE

Se si vuole definire una frase con un numero variabile di parole, è possibile inserire una derivazione nel diagramma. Se si definisce:

la grammatica

darà

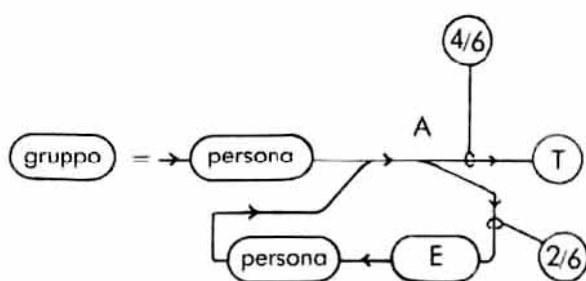
JOHN POVERO
ANNA GRASSA
KATE
o TOM POVERO

poichè il guidatore del tram può scegliere a caso se seguire la strada dell'aggettivo o no. Le grammatiche a linea tranviaria possono contenere delle iterazioni. Si consideri il diagramma

Ricordarsi che il guidatore ha la libera scelta quando arriva al punto A nel diagramma. Se va diritto, raggiungerà il capolinea e terminerà la frase. Se gira a destra aggiungerà un'altra persona. Alcune delle frasi che egli può produrre, sono

TOM
TOM INTELLIGENTE E KATE
TOM GRASSO E ANNA POVERA
E JOHN INTELLIGENTE
TOM E JOHN E KATE POVERA
oppure TOM GRASSO E TOM GRASSO
E TOM GRASSO

Sotto alcuni aspetti il diagramma a linee tranviarie è come uno schema di flusso. La differenza essenziale è che nei punti di scambio tipo A, la scelta del binario avviene casualmente e non in risposta ad una specifica domanda. L'elemento casuale è essenziale altrimenti il diagramma produrrebbe la stessa frase ogni volta.

PROBABILITÀ

Quantunque la scelta del percorso non sia determinata in anticipo, si vuole sempre mantenere tuttavia un certo tipo di controllo su di essa, altrimenti il guidatore potrebbe decidere di continuare l'iterazione all'infinito. È possibile fare ciò aggiungendo una probabilità o una possibilità a ciascuno dei percorsi possibili.
Un modo per far ciò consiste nel dare al guidatore un dado a sei facce e istruirlo a lanciarlo ogniqualvolta deve compiere una scelta. Nel punto A, per esempio, gli diremo di girare a destra se lanciando ottiene un 5 o un 6, ma di procedere fino al capolinea se ottiene un 1, un 2, un 3 o un 4. Ciò significa, nel lungo termine, che egli può prevedere di girare a destra due volte ogni sei volte che passa per A e procedere per quattro volte. Possiamo indicare questo sul diagramma aggiungendo dei marcatori di probabilità come questi:

Notare che le probabilità dei percorsi che traggono origine da un punto tipo A devono dare per somma 1, ossia la certezza, in quanto il guidatore del tram è costretto a prenderne in ogni caso uno. Se si considerano i marcatori di probabilità come frazioni, la loro somma deve dare 1. Nella definizione di (gruppo) essi corrispondono a:
4/6 + 2/6 = 6/6 = 1.
Si consideri ora come il **64** può essere messo in condizioni di produrre frasi casuali.
Le regole di base sono le seguenti:

1) La frase che viene costruita viene contenuta nella variabile stringa X1$. La variabile inizia come stringa nulla e le parole sono aggiunte una alla volta. Ciascuna parola è preceduta da uno spazio.

2) Ogni diagramma di grammatica separato è rappresentato da una subroutine. Uno dei suoi parametri, sia per l'input che per l'output, è X1$.

Osserviamo ora qualche operazione elementare. Per aggiungere un nome noto alla frase, dobbiamo semplicemente concatenarlo come segue:

```
X1$=X1$ + " E"
```

Per scegliere una parola casuale da una lista, il metodo più semplice è di assicurarsi che tutte le parole candidate si trovino in una matrice. Si supponga che ci siano J di esse in elementi consecutivi iniziando da N$(K) per terminare in N$(K+J−1). Quindi, il seguente comando ne preleverà una a caso e la collegherà a X1$:

```
X1$=X1$+" "+N$(K+INT(J★RND(0)))
```

Ciò funziona in quanto l'espressione indice K+INT(J★RND(0)) ha la stessa probabilità di uscire con qualsiasi numero tra K e K+J−1: esattamente ciò che ci serve.
Per eseguire un percorso di linea tranviaria con una data probabilità, possiamo usare la condizione

RND(0)<p/q
(dove p/q è il marcatore di probabilità)

Se poniamo

RND(0)<4/6

la condizione sarà vera quattro volte su sei e falsa le altre due volte. Ciò significa che l'altro marcatore di probabilità (2/6) non deve essere scritto nel programma.
Possiamo ora costruire un programma per produrre frasi di gruppi così come sono definite dalla nostra grammatica. Iniziamo impostando una matrice con i nomi e gli aggettivi e inizializziamo X1$ ad un valore nullo. Ciò occupa le righe da 10 a 40 nel programma che segue.
Successivamente, scriviamo una subroutine per ciascuno dei diagrammi a linea tranviaria. Quello che inizia alla riga 1000 è per un (nome). Le costanti nell'espressione indice sono 1 e 4 in quanto ci sono 4 nomi possibili che iniziano in N$(1). Analogamente, la subroutine che inizia in 1100 produce un (aggettivo) e le appropriate costanti nelle espressioni di indice sono 5 e 3.

La subroutine in 1200 dà una (persona). Stabiliamo la probabilità di usare un (aggettivo) 3/6 — il che indica che la probabilità di non averne una è pure 3/6: probabilità pari.
A 1300 troveremo la subroutine per il diagramma (gruppo). Notare con quanta precisione segue le linee tranviarie:

1310 seleziona sostantivo (persona)

1320 produce una decisione a caso per finire la frase

1330 aggiunge la congiunzione (E)

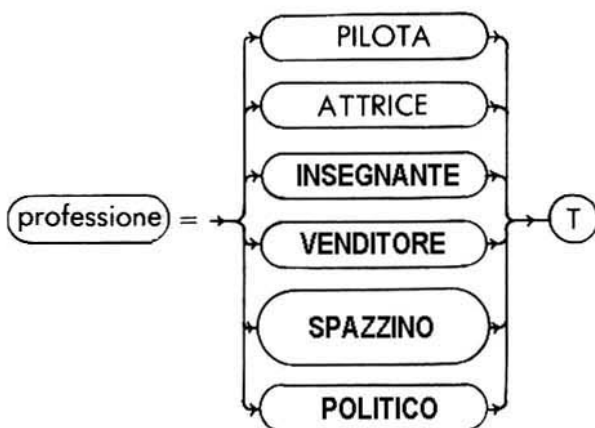
1340 seleziona un'altra (persona)

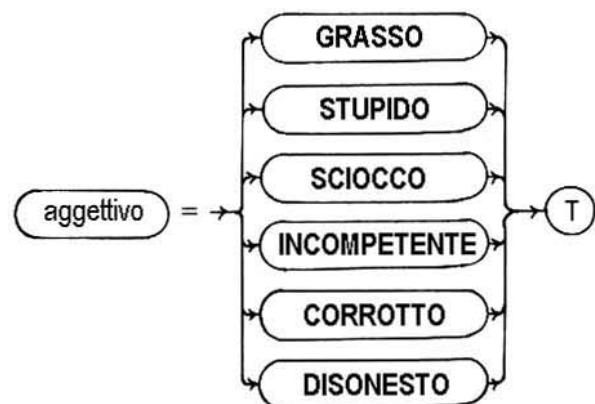
1350 rimanda alla punto della subroutine dove decide di finire o andare a ripetere

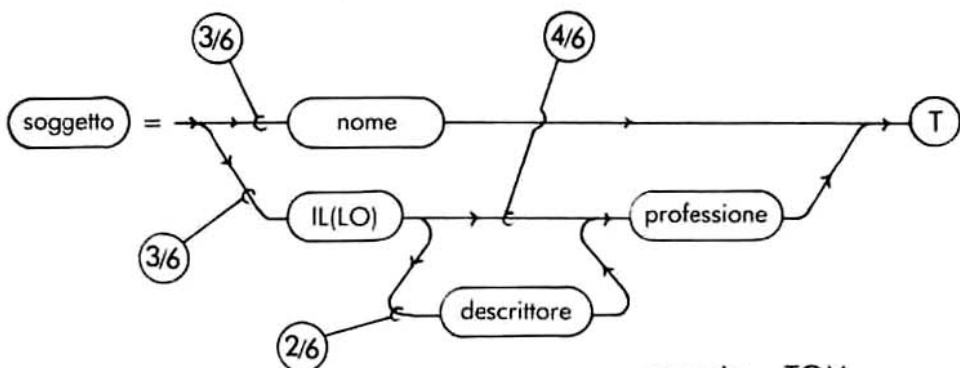
Finalmente, forniamo qualche comando "guida" nelle linee 40, 50 e 60.


```
10 DIM N$(7)
20 N$(1) = "TOM ": N$(2)="KATE ":
   N$(3)="JOHN ":N$(4)="ANNE "
30 N$(5) = " GRASSO": N$(6)="ESPERTO":
   N$(7)="POVERO"
40 X1$=""
50 GOSUB 1300
60 PRINT X1$
70 GOTO 40

1000 REM FIRST NAME
1010 X1$=X1$+" "+N$(1+INT(4★RND(0)))
1020 RETURN

1100 REM ADJECTIVE
1110 X1$=X1$+" "+N$(5+INT(3★RND(0)))
1120 RETURN

1200 REM PERSON
1210 IF RND(0) < 3/6 THEN GOSUB 1100:
     REM CALL ADJECTIVE
1220 GOSUB 1000 : REM CALL PERSON
1230 RETURN

1300 REM GROUP
1310 GOSUB 1200 : REM CALL PERSON
1320 IF RND(0) < 4/6 THEN RETURN : REM
     POINT "A" IN DIAGRAM
1330 X1$=X1$+" AND "
1340 GOSUB 1200 : REM CALL ANOTHER
     PERSON
1350 GOTO 1320
```

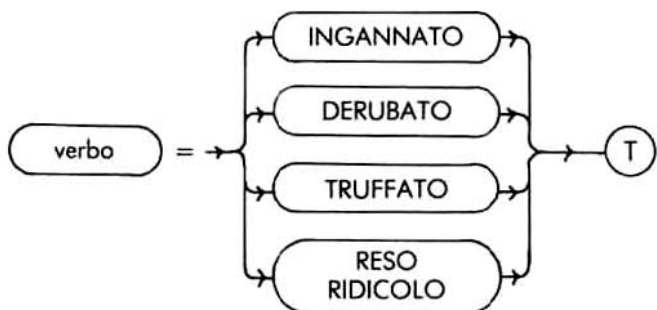
Batti questo programma e fallo girare. Poi prova l'effetto di variare le probabilità nelle linee 1210 e 1320.

Una volta che siano stati stabiliti i principi di costruire frasi a caso, si possono facilmente estendere per completare le frasi. Studiare le seguenti definizioni, e scrivere gli esempi delle frasi o delle sentenze che si vuole produrre:

es. **VENDITORE**

o ....

es. **SCIOCCO**

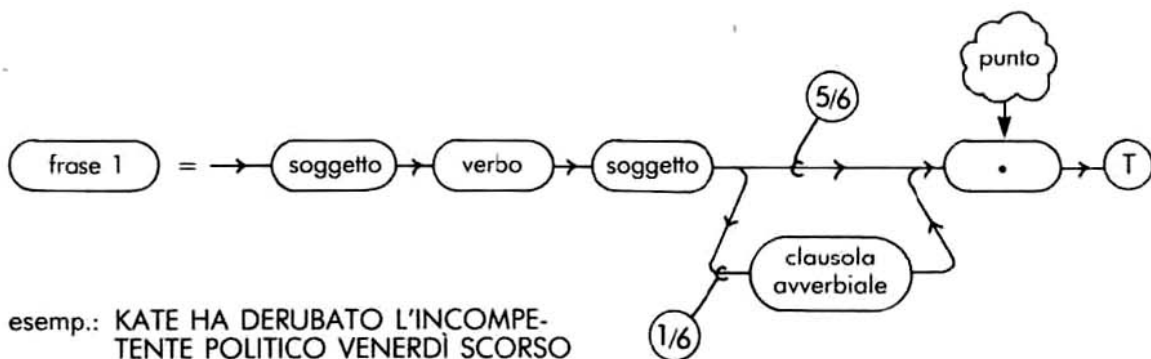
o ....

esempio TOM

o LO STUPIDO PILOTA

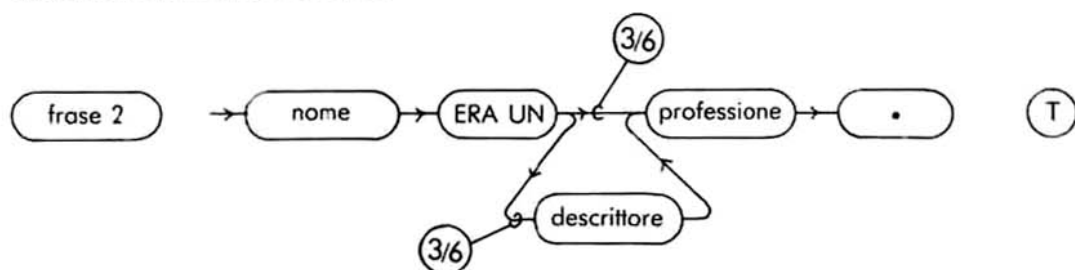
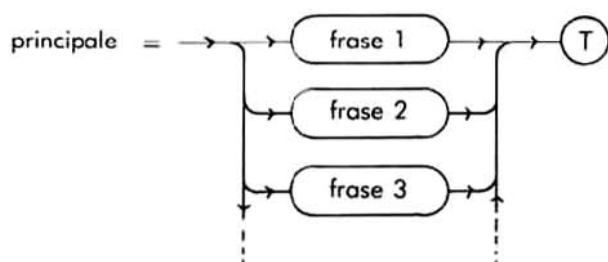
o . . . .

esempio DERUBATO

oppure . . . .

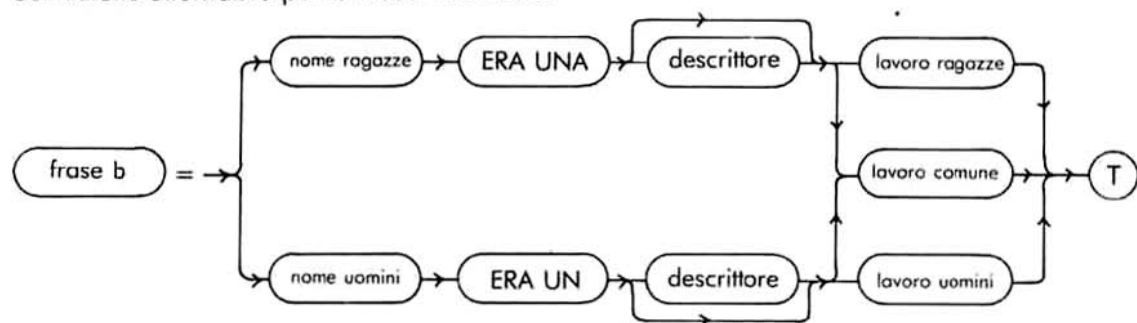
esempio IERI

oppure . . . . .

esemp.: KATE HA DERUBATO L'INCOMPETENTE POLITICO VENERDÌ SCORSO

oppure . . . .

È possibile creare qualsiasi numero di frasi con diverse definizioni, ad esempio:

ed è possibile combinarle in un diagramma a linea tranviaria (principale) che comprende tutte le forme di frasi che si vogliono generare. Potrebbe cominciare:

A questo punto vale la pena di osservare il raggruppamento di parole. Nella serie attuale di diagrammi e linee tranviarie, una frase possibile è

TOM ERA UN ATTRICE PIGRA.

Per evitare questo tipo di assurdità, occorre separare i nomi in due gruppi e le professioni in tre: quelli limitati agli uomini (ad esempio VESCOVO), quelli limitati alle donne (ad esempio ATTRICE) e quelli aperti ad entrambi. Una definizione alternativa per la frase 2 sarebbe:

Per completare questo esempio, dovremmo consigliare alcuni dettagli pratici. Innanzitutto le frasi prodotte dal sistema dovrebbero essere correttamente disposte e ciò può essere fatto dalla subroutine descritta nell'Unità 21.
Secondo, il programma diventa più interessante per gli utenti se essi possono fornire proprie liste di parole per le varie categorie — probabilmente alterando le istruzioni DATA nel programma dopo che è stato caricato da una cassetta. Ciò implica che il programmatore non conosca nè le parole nè quante ce ne saranno in ciascun gruppo. Ci saranno chiaramente difficoltà nello scrivere espressioni indice adatte.

Questo problema può essere superato facendo in modo che il programma definisca una serie di "cartelli indicatori" ai vari gruppi di parole.
Nell'attuale grammatica, abbiamo dei gruppi di parole: nomi, aggettivi, professioni, descrittori, verbi e clausole avverbiali. Diciamo all'utente di inserire la sua scelta per ciascun gruppo in una (o più) istruzioni DATA e di terminare con una "Z". L'utente potrebbe inserire:

```
10 DATA RONALD,JIMMY,GERALD,
        RICHARD,Z
20 DATA GRAZIOSO,BRILLANTE,SOAVE,Z
30 DATA POLITICO,PROGRAMMATORE,
        SUONATORE,VENDITORE,
        BRICCONE,Z
40 DATA MALVAGIO,PROFESSIONALE,
        ESPERTO,Z
        50 DATA INCONTRATO,AIUTA-
        TO,
        SALUTATO,Z
60 DATA IERI SERA, NEL PARCO, PRIMA
        DI ESSERE UCCISO,Z
```

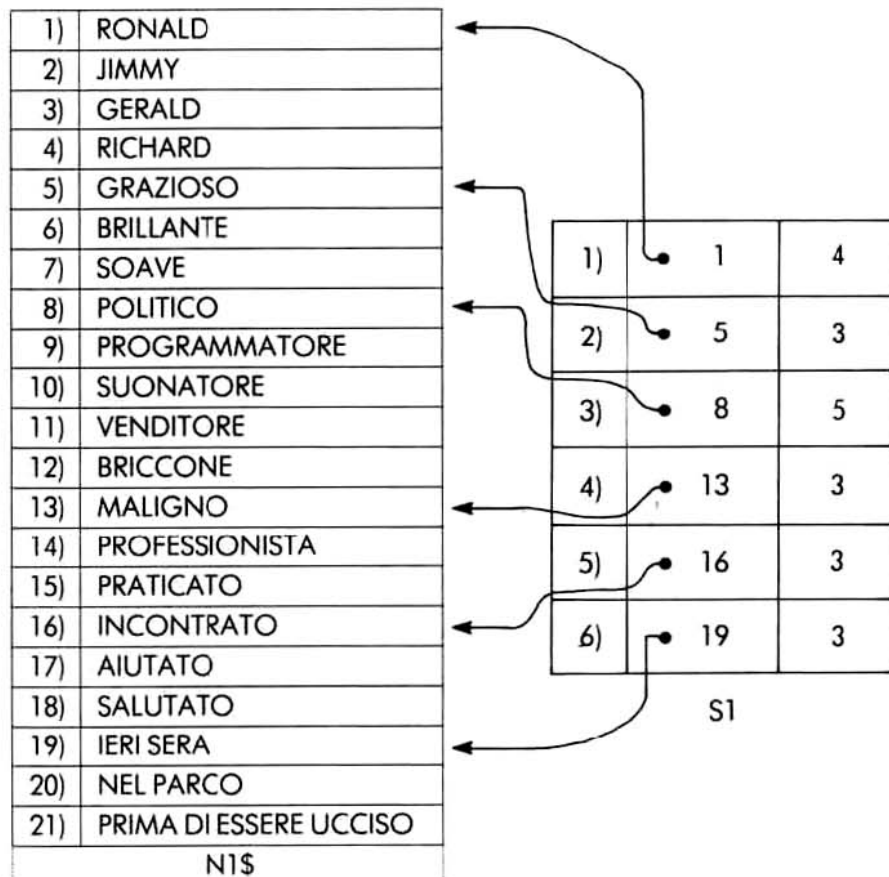

All'interno del programma disponiamo i dati in matrici: una con un elemento per ciascuna parola (salvo le Z) e una con le informazioni su ciascun gruppo. Le necessarie informazioni comprendono

a) La posizione iniziale (e cioè l'indice del primo elemento) nel gruppo

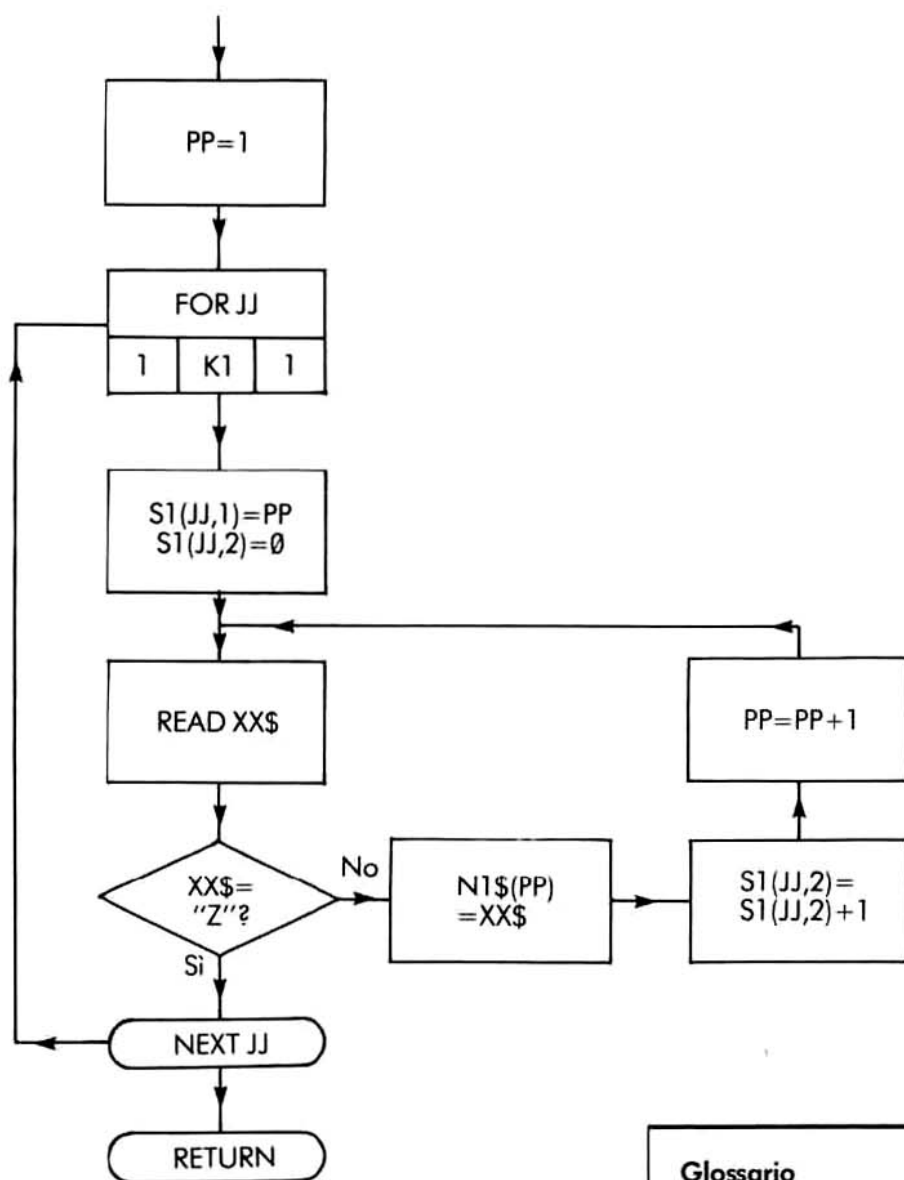
b) il numero di parole nel gruppo.

Un diagramma potrebbe aiutare a rendere tutto più chiaro:

	N1$
1)	RONALD
2)	JIMMY
3)	GERALD
4)	RICHARD
5)	GRAZIOSO
6)	BRILLANTE
7)	SOAVE
8)	POLITICO
9)	PROGRAMMATORE
10)	SUONATORE
11)	VENDITORE
12)	BRICCONE
13)	MALIGNO
14)	PROFESSIONISTA
15)	PRATICATO
16)	INCONTRATO
17)	AIUTATO
18)	SALUTATO
19)	IERI SERA
20)	NEL PARCO
21)	PRIMA DI ESSERE UCCISO

	S1	
1)	1	4
2)	5	3
3)	8	5
4)	13	3
5)	16	3
6)	19	3

In questa struttura di dati, la fila 3 della matrice S1 dice al programma che il gruppo 3 contiene 5 parole che iniziano in N1$ (8). Un'istruzione per aggiungere una parola del gruppo 3 a X1$ verrebbe letta così

$$X1\$=X1\$ + "" + N1\$(\underbrace{S1(3,1)}_{=8} + INT(\underbrace{S1(3,2)}_{=5} \star RDN(\emptyset)))$$

Posto che i cartelli indicatori in S1 siano correttamente disposti, questa espressione funzionerà per qualsiasi raccolta di parole che l'utente fornisca.
La messa a punto dei cartelli indicatori è illustrata nel seguente schema di flusso:

Glossario

N1$: Matrice per le parole
S1: Matrice per i cartelli indicatori
K1: Numero dei gruppi
XX$: Parola corrente
JJ: Contatore di gruppo
PP: Contatore di parola

La specifica e il codice corrispondenti sono:

Specifica della subroutine

Scopo: Leggere gruppi di parole per generare frasi casuali.

Righe: 5000-5070

Parametri: *Output:* N1$ (parole), S1 (cartelli indicatori

Matrici vuote; K1: Numero di gruppi

Output: N1$, S1 (impostato come descritto nel testo)

Locali: PP,JJ,XX$

```
5000 REM LEGGI PAROLE E IMPOSTA
     SEGNAPOSTI
5010 PP=1
5020 FOR JJ = TO K1
5030 S1(JJ,1)=PP:S1(JJ,2) = 0
5040 READ XX$
5050 IF XX$<>"Z"THEN N1$(PP)=XX$:
     S1(JJ,2)=S1(JJ,2)+1:PP=PP+1:
     GOTO 5040
5060 NEXT JJ
5070 RETURN
```

Per riassumere: il programma per generare le frasi che abbiamo discusso si compone principalmente di subroutine che sono abbastanza strettamente modellate sulla grammatica delle frasi da costruire. In questo modo la struttura del problema è trasferita pressochè senza alterazione al programma stesso.

ESPERIMENTO 25·1

Scrivere un generatore di frasi complete che comporta parecchi tipi di frasi. Provarlo con gli amici e i parenti.

Esperimento 25.1 completato	

GIOCHI DI AVVENTURA O DI LABIRINTI

Nel successivo esempio pratico esamineremo due esempi in cui la struttura del problema è riflessa nei dati anzichè nel programma.
Esistono numerosi giochi per computer in cui l'eroe deve esplorare un castello/labirinto/universo, affrontare vari pericoli tipo dragoni o alieni e salvare una principessa/un modulatore ipertermico/e un intrepido esploratore dello spazio. Il disegno di una semplice versione di uno di tali giochi può essere esposto come un diagramma del tipo illustrato in Fig. 25.2. Quando si inizia la codifica di un tale gioco, il modo più naturale consiste nel partire scrivendo mucchi di codici senza profilo tipo questo

```
10 PRINT " [SHIFT] e [CLR HOME] ";
20 PRINT "LA MISSIONE È DI RECUPERARE"
...
100 PRINT" OR B) ASPETTI E OSSERVI?"
110 INPUT X$
120 IF X$ <> "A" AND X$ <> "B" THEN
    PRINT "PROVA ANCORA": GOTO 110
130 IF X$ = "A" THEN 500
140 PRINT " [SHIFT] e [CLR HOME] ";
150 PRINT "SEI ATTACCATO DA"
...

...
270 PRINT OR B) FUGGI NEL BATTELLO DI
    SALVATAGGIO
280 INPUT X$
290 IF X$ <>"A" AND X$ <> "B" THEN
    PRINT "PROVA  ANCORA": GOTO 280
300 IF X$ = "A" THEN 500

...
```

e così di seguito

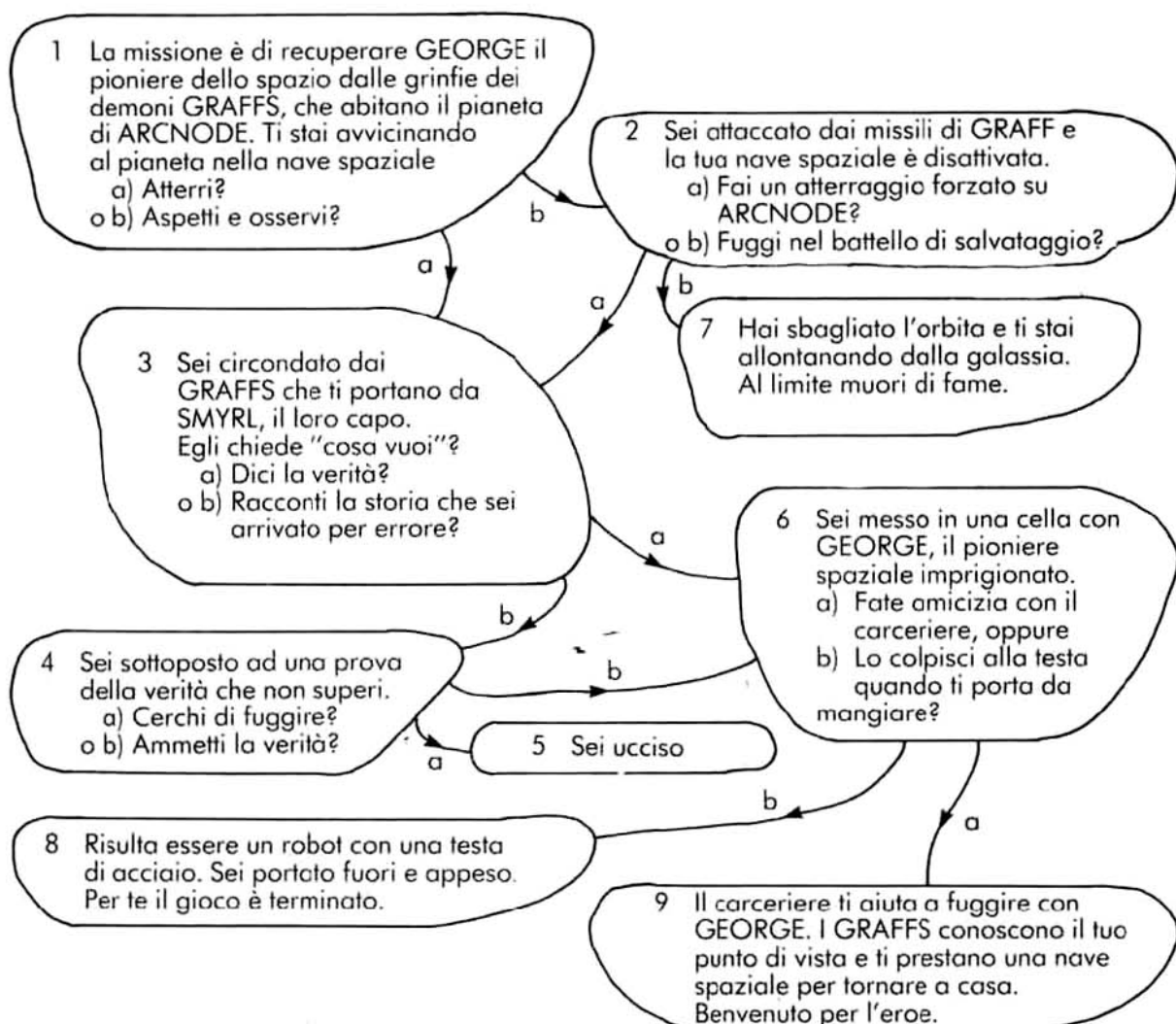



Figura 25.2

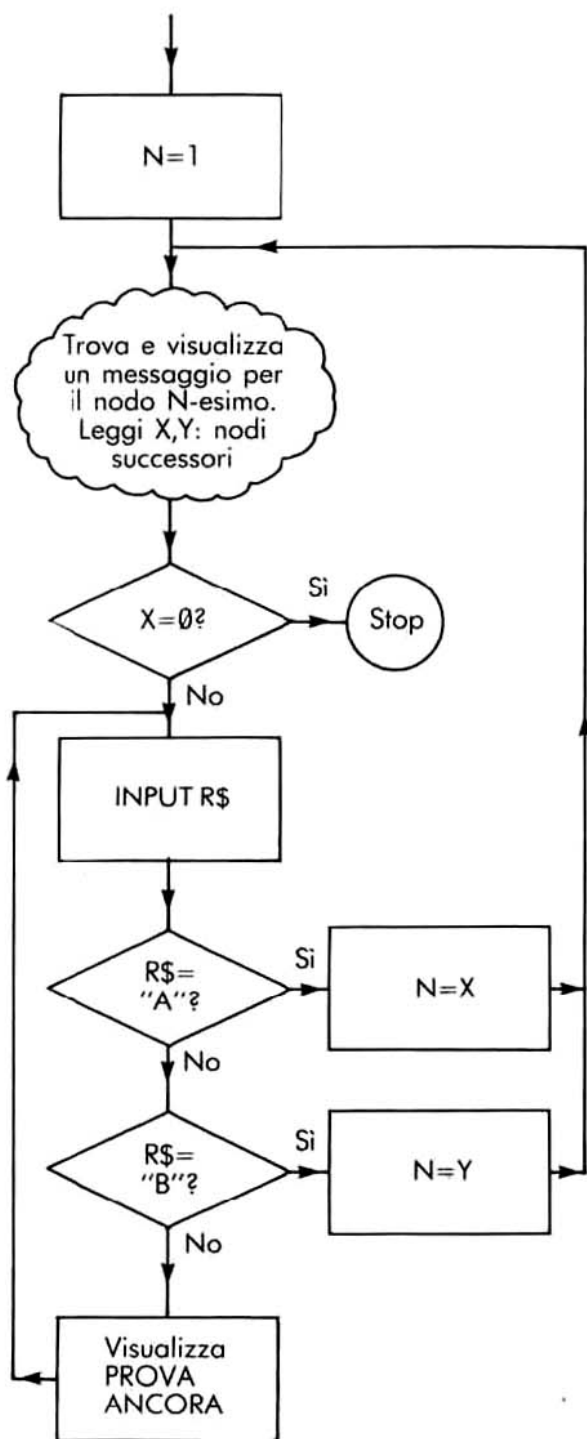
Questo programma è difficile da leggere e da modificare e si sviluppa rapidamente per occupare tutto lo spazio nella memoria.
Diamo uno sguardo a ciò che succede in ciascun punto o nodo nel diagramma. Ci sono due possibilità:

1) Il computer cancella lo schermo e visualizza un messaggio. Quindi interrompe il programma. Ciò succede quando l'eroe viene ucciso oppure quando riesce nella sua missione.

2) a) Il computer cancella lo schermo e visualizza un messaggio.

   b) Quindi invita l'utente a battere A o B e respinge tutti gli altri input.

   c) Usa la nuova risposta per scegliere un nuovo successore e ripete il processo da capo.

Ciò suggerisce che il gioco può essere gestito da un programma molto semplice, con tutta la complessità racchiusa nei dati.
Si supponga di numerare i nodi da 1 in su (come già è stato fatto nel diagramma). Quindi, per ciascun nodo, possiamo creare una "package" di dati composto dalla stringa da visualizzare e dai numeri dei due nodi successivi. Usiamo la convenzione secondo cui un numero successore di 0 significa che il gioco è terminato.
Lo schema base di flusso per il programma è ora abbastanza breve. Esso è:

Glossario:

N: Numero corrente di nodo
X,Y: Numeri dei nodi successori
R$: Risposta

Qui di seguito è indicato il codice per il programma. Notare che il messaggio per ciascun nodo è solitamente troppo lungo per essere considerato come un singolo elemento di dati. Usiamo quattro elementi che consentono un testo di un massimo di 240 caratteri per nodo. Degli elementi, i primi due descrivono la nuova situazione e gli altri danno i corsi di azione probabile.
La subroutine 5500 è quella data dall'Unità 21, per visualizzare le frasi senza separare le parole.

```
10 DATA " SHIFT and CLR HOME YOUR
    MISSION IS TO RESCUE GEORGE
    THE SPACE PIONEER FROM THE
    CLUTCHES OF THE "
11 DATA " GRAFFS, WHO INHABIT THE
    PLANET ARCNODE, DO YOU"
12 DATA " A) LAND","OR B) WAIT AND
    WATCH",3,2
...
```


seguiti da simili gruppi per ciascuno degli altri nodi. Ciascun gruppo contiene quattro stringhe e 2 numeri.

```
1000 REM PROGRAM BEGINS HERE
1010 N=1 : REM START AT NODE 1
1020 RESTORE : REM FIND N'TH NODE
1030 FOR J = 1 TO N
1040 READ J$,K$,L$,M$,X,Y : REM AND
     READ ITS CONTENTS
1050 NEXT J
1060 X1$=J$ + K$ : GOSUB 5500: REM
     DISPLAY MESSAGE
1070 X1$ = L$ : GOSUB 5500 :REM FIRST
     ALTERNATIVE
1080 X1$ = M$ : GOSUB 5500 :REM
     SECOND ALTERNATIVE
1090 IF X = 0 THEN STOP : REM END OF
     GAME
1100 INPUT R$ : REM GET REPLY
1110 IF R$ = "A" THEN N=X : GOTO 1020
1120 IF R$ = "B" THEN N=Y : GOTO 1020
1130 PRINT "TRY AGAIN": GOTO 1100
```

seguito dai comandi della subroutine in 5500.

ESPERIMENTO 25.2

Caricare l'intero programma di GRAFFS dalla cassetta di nastro e provarlo. Quindi modificarlo:

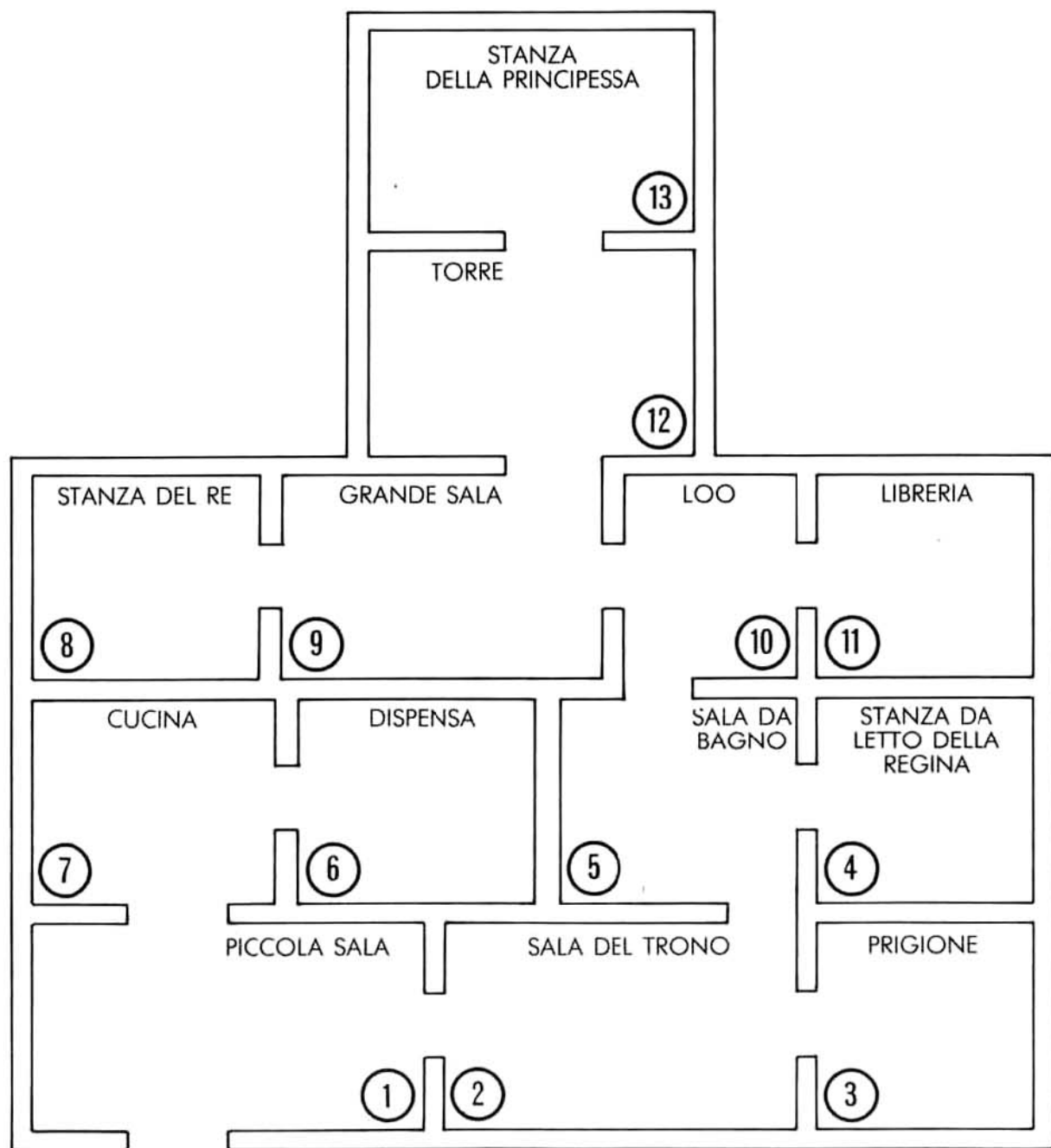
a) Per produrre interessanti effetti sonori

b) Per raccontare una storia diversa.

Esperimento 25.2 completato	

ESPERIMENTO 25·3

Il programma GRAFFS del precedente esperimento ha offerto un semplice esempio del modo in cui la struttura di un problema può essere rappresentata dai dati. Un esempio più complesso è dato dal programma denominato DUNGEON, anche esso caricabile da cassetta ed eseguibile. Questo programma segue un profilo popolare, ma non è altrettanto ambiguo come i suoi equivalenti commerciali in quanto deve essere sufficientemente breve per poterlo studiare in dettaglio.
Una volta giocata questa versione di DUNGEON alcune volte, esaminare il codice attentamente e costruire propri schemi di flusso. Potrebbero essere utili le seguenti informazioni:

1) Pianta del castello:

2) Codice interno:

Stanze

Esterno	0
Piccola sala	1
Sala del trono	2
Prigione	3
Stanza da letto della regina	4
Sala da bagno	5
Dispensa	6
Cucina	7
Stanza del re	8
Grande sala	9
Loo	10
Libreria	11
Torre	12
Stanza della principessa	13

Pericoli

Dragone	1
Ragno	2
Vespe	3
Mago	4
Pescecane	5

Armi

Spada	1
Bastone	2
Bomboletta spray	3
Pozione magica	4
Lanciafiamme	5

Il pericolo numero X può essere soltanto superato con l'arma X.

Glossario

n: Numero delle stanze

mm$(5,5): Testo per le battaglie con ciascuno dei cinque pericoli. Pertanto da mm$(1,1) fino a mm$(1,5) contengono i messaggi per i combattimenti con il dragone:
"a Dragon" (un dragone)
"You fight and" (combatti e)
"kill with your sword" (lo uccidi con la spada)
"it kills you" (lui t'uccide)
"You both run away!" (entrambi fuggite)
La subroutine della battaglia (dalla riga 3000 in su) usa questi messaggi per visualizzare i risultati delle battaglie.

w$(5): I nomi delle cinque armi.

r$(n): I nomi delle stanze nel castello.

c%(n,4): Le vie attraverso le quali le stanze sono collegate. Per esempio, la fila da c%(2,1) a c%(2,4) è 5,3,0,1. Ciò significa che la sala del trono (2) è collegata alla sala da bagno (5) andando verso nord, la prigione (3) andando verso ovest e la piccola sala (3) andando verso est. Non c'è porta a sud nella sala del trono.

di$(4): I nomi dei quattro punti cardinali: Nord, Est, Sud e Ovest.

d%(5): Le posizioni presenti (come numeri di stanze) dei cinque pericoli. Pertanto se d%(2) = 7, ciò significa che il ragno è in cucina!

l%(3): Le armi che l'eroe ha con se.

w%(5): Le posizioni delle armi che l'eroe ha perso (o non ha ancora preso).

p: Posizione della principessa (come numero di stanza).

h: Posizione dell'eroe (come numero di stanza).

q: 1 se la principessa è con l'eroe. 0 se non l'ha ancora trovata (oppure se l'ha abbandonata).

Infine, quando finalmente si è capito come funziona il programma DUNGEON, disegnare e scrivere un proprio programma sulle stesse linee generali.

Esperimento 25.3 completato	

UNITA':26

Gli Sprites	PAGINA 287
Esperimento 26.1	289
Disegno degli Sprites	289
Esperimento 26.2	291
Inserimento degli Sprites in programma	291
Il controllo degli Sprites	292
Esperimento 26.3	296
Studio di 4 programmi Sprite	296
Esperimento 26.4	297
Esperimento 26.5	300
Esperimento 26.6	301
Sprites multicolori	301
Schermi con molti Sprites	302

GLI SPRITES

Questa unità si occupa del disegno e dell'uso degli sprites. Uno Sprite è una piccola immagine che si muove sullo schermo dolcemente, indipendentemente da qualsiasi altra cosa che può essere visualizzata contemporaneamente. Come si ricorderà, il programma dimostrativo fornito insieme alla Parte 1 di questa serie, mostrava alcune immagini di fuochi d'artificio, palloni e paracadute. Queste immagini non erano che altrettanti esempi di Sprites.

La funzione Sprite consente di presentare vivaci immagini animate per illustrare il programma. Gli Sprite sono preziosissimi in molti settori: giochi, programmi educativi e sorveglianza e controllo industriale.

Si inizierà raccomandando un poco di attenzione. Gli Sprites non sono facili da manipolare, ma non sono neppure particolarmente difficili. Occorre in ogni caso esercitare tutte le proprie abilità artistiche nonché occorre avere cura e pazienza nella programmazione se si vuole ottenere un effetto realmente impressionante.

ESPERIMENTO 26•1

Per iniziare, caricare ed eseguire lo Sprite Display Program, SDP. Provare alcune delle cose che esso può fare:

a) Quando si premono i tasti del cursore (e SHIFT) nel modo normale, il pesce si muove verso l'alto e verso il basso, verso sinistra e verso destra. Può anche muoversi e uscire dallo schermo. È possibile scegliere la velocità di movimento battendo F ("Fast" per veloce) o S ("Slow" per lento).

Notare che quando il pesce si muove lentamente, si muove anche uniformemente - non a scatti in corrispondenza di caratteri o di linee.

b) Il pesce cambierà forma e dimensione quando si batte ↓ o ←. In questo programma, questi testi hanno un doppio effetto: cioè battuti per la seconda volta, invertono l'effetto della prima.

c) Gli otto tasti dei colori, quando usati insieme al tasto CTRL o al tasto C=, colorano il pesce in uno dei sedici colori disponibili. Uno di essi, in grigio chiaro, è invisibile contro il colore di fondo cosicché compare un messaggio di avvertimento in quanto il pesce è perfettamente invisibile.

d) Se si sposta il pesce alla parte dello schermo in cui sono visualizzate le istruzioni, lo si vedrà apparentemente andare dietro alle lettere rimanendo peraltro perfettamente visibile.

Battere ora il tasto £.

Ciò porta il pesce in avanti, in modo che si trovi davanti ai caratteri e li nasconda.

Battere di nuovo il tasto £ per riportare il pesce dietro i caratteri.

Questa è una dimostrazione della funzione "Data-Sprite Priority". Gli Sprites e i caratteri visualizzati sono normalmente indipendenti ma se si devono mostrare oggetti nello stesso punto, il computer deve decidere che mettere davanti. Fortunatamente questa decisione è sotto controllo del programma. Pensare per un attimo cosa succederebbe se si ripetesse questo esperimento con il pesce nel colore "invisibile". Quindi provare se l'effetto previsto si è realmente verificato.

Esperimento 26.1 Completato	

Il programma "pesce" ha presentato alcune delle proprietà più semplici degli Sprites. Qui di seguito compare un listato completo del programma al quale si dovrebbe fare riferimento studiando questa unità.

```
10 REM COPYRIGHT (C) ANDREW COLIN
   1983
15 REM
20 REM DEFINITION OF FISH SPRITE
30 DATA0,8,0,0,48,0,0,192,0
40 DATA3,128,3,31,192,28,115,248,112
50 DATA243,255,224,255,255,192,127,
   255,224
60 DATA31,248,112,7,224,28,1,192,3
70 DATA0,224,0,0,24,0,0,4,0
80 DATA0,0,0,0,0,0,0,0,0
90 DATA0,0,0,0,0,0,0,0,0
100 REM
110 REM SET UP SPRITE DATA
120 FORJ=832 TO 894
130 READ A: POKE J,A
140 NEXT J
150 REM
160 REM SET V = ADDRESS OF SPRITE
    CONTROLLER
170 V=208*256
180 REM DISPLAY INSTRUCTIONS
185 POKE 53281,15
190 PRINT " [SHIFT] and [CLR HOME] [CTRL]
    and [! 1] [CRSR ↑↓] "
200 PRINT"     SPRITE DEMONSTRATION
    PROGRAM"
205 PRINT"     ====== ========
    ===== ======="
210 PRINT:PRINT
220 PRINT" USE CURSOR CONTROLS TO
    MOVE"
225 PRINT" THE FISH —"
230 PRINT
240 PRINT" COLOUR KEYS"
245 PRINT" (AND CRTL OR COMMODORE
    KEY)"
250 PRINT" TO CHANGE ITS COLOUR "
260 PRINT
270 PRINT" ↑ AND ← TO CHANGE ITS
    SHAPE"
271 PRINT:PRINT" F OR S TO MAKE IT
    MOVE "
272 PRINT" FAST OR SLOW"
275 PRINT:PRINT" AND £ TO PUT IT IN
    FRONT OF"
278 PRINT" OR BEHIND THE TEXT ON THE
    SCREEN"
280 REM
290 REM SET UP THE SPRITE FOR DISPLAY
300 POKE 2040,13 : REM SET POINTER
    TO SPRITE DEFINITION (13*64=832)
310 X=200:Y=230 :R=1:REM SET SPRITE
    POSITION AND RATE
315 POKE V+39,1:POKE V+27,1:REM
    SELECT COLOUR WHITE; SPRITE IN
    FRONT OF TEXT
320 POKE V+23,0:POKE V+29,0:REM
    SELECT SMALL DIMENSIONS
330 POKE V+21,1:REM ENABLE SPRITE
340 REM UPDATE SPRITE POSITION AND
    READ KEYBOARD
350 POKE V,X AND 255:REM LOAD
    HORIZONTAL POSITION
360 POKE V+16,INT(X/256)
370 POKE V+1,Y: REM SET VERTICAL
    POSITION
380 GET A$:IF A$="" THEN 380
385 PRINT" [CLR HOME] [SPACE]   30 times"
390 REM ANALYSE KEY PRESSED
400 IF A$ = " [CRSR ⇐⇒] " AND X<343 THEN
    X=X+R:GOTO340:REM MOVE RIGHT
410 IF A$ = " [SHIFT] and [CRSR ⇐⇒] " AND
    X>0 THEN X=X-R:GOTO 340:REM
    MOVE LEFT
420 IF A$ = " [CRSR ↑↓] " AND Y<249 THEN
    Y=Y+R:GOTO340:REM MOVE
    DOWN
440 IF A$ = " [SHIFT] and [CRSR ↑↓] " AND
    Y>8 THEN Y=Y-R:GOTO340:REM
    MOVE UP
450 IF A$ = "↑" THEN POKE V+23,1-PEEK
    (V+23):GOTO340 : REM CHANGE
    VERTICAL SIZE
460 IF A$ = "←" THEN POKE
V+29,1-PEEK
    (V+29):GOTO340 : REM CHANGE
    HORIZONTAL SIZE
470 REM NOW TEST COLOUR KEYS
480 IF A$ = " [CTRL] and [! 1] "THEN
    POKE V+39,0:GOTO340:REM BLACK
490 IF A$ = " [CTRL] and [" 2] "THEN
    POKE V+39,1:GOTO340:REM WHITE
500 IF A$ = " [CTRL] and [# 3] "THEN
    POKE V+39,2:GOTO340:REM RED
510 IF A$ = " [CTRL] and [$ 4] "THEN
    POKE V+39,3:GOTO340:REM CYAN
520 IF A$ = " [CTRL] and [% 5] "THEN
    POKE V+39,4:GOTO340:REM PURPLE
530 IF A$ = " [CTRL] and [& 6] "THEN
    POKE V+39,5:GOTO340:REM GREEN
540 IF A$ = " [CTRL] and [' 7] "THEN
    POKE V+39,6:GOTO340:REM BLUE
550 IF A$ = " [CTRL] and [( 8] "THEN
    POKE V+39,7:GOTO340:REM
    YELLOW
560 IF A$ = " [C=] and [! 1] "THEN
    POKE V+39,8:GOTO340:REM
    ORANGE
570 IF A$ = " [C=] and [" 2] "THEN
```

```
        POKE V+39,9:GOTO34Ø:REM BROWN
580 IF A$ = " [C=] and [#3] "THEN
        POKE V+39,1Ø:GOTO34Ø:REM PINK
590 IF A$ = " [C=] and [$4] "THEN
        POKE V+39,11:GOTO34Ø:REM DARK
        GREY
600 IF A$ = " [C=] and [%5] "THEN
        POKE V+39,12:GOTO34Ø:REM
        MEDIUM GREY
610 IF A$ = " [C=] and [&6] "THEN
        POKE V+39,13:GOTO34Ø:REM LIGHT
        GREEN
620 IF A$ = " [C=] and ['7] "THEN
        POKE V+39,14:GOTO34Ø:REM LIGHT
        BLUE
630 IF A$ = " [C=] and [(8] "THEN
        POKE V+39,15:GOTO34Ø:PRINT"
        THE FISH IS NOW INVISIBLE":GOTO34Ø
640 IF A$="£" THEN POKE V+27,1—PEEK
        (V+27):GOTO 34Ø:REM CHANGE
        PRIORITY
650 IF A$="F" THEN R=5:GOTO34Ø:
        REM GO FAST
660 IF A$="S" THEN R=1:GOTO 34Ø:
        REM GO SLOW
800 GOTO34Ø
```

Ci sono tre cose importanti da imparare a proposito degli Sprites:

a) Come disegnarli
b) Come inserirli nel programma
c) Come controllarli sullo schermo

DISEGNO DEGLI SPRITES

Il "disegno degli Sprites" è sostanzialmente un esercizio artistico. Ogni Sprite si compone di un insieme di punti luminosi detti *pixel*, disegnati entro un rettangolo di 21 righe per 24 colonne. Per cominciare occorre munirsi di carta quadrettata e disegnare una griglia di questo tipo.

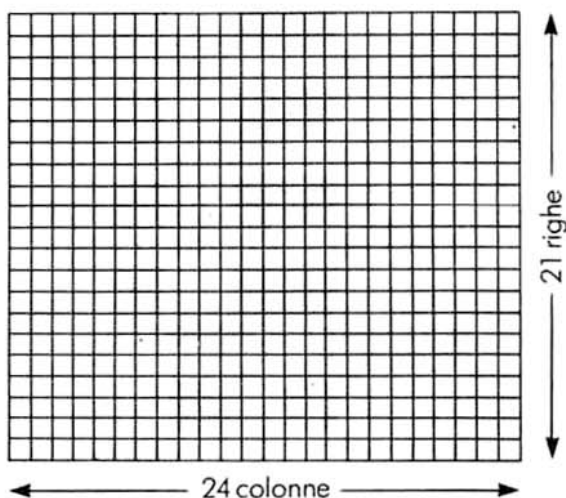



Figura 26.1

Disegnare ora lo Sprite sulla griglia, usando penna e *gomma*. Funzionano ugualmente bene disegni "pieni" e "profili". Si finirà per avere un disegno *digitalizzato* - in cui ogni quadratino o pixel è completamente bianco o completamente nero. Il risultato sembra abbastanza grezzo quando lo si osserva sulla griglia ma occorre ricordare che lo Sprite sullo schermo sarà molto più piccolo del disegno e pertanto i suoi difetti non saranno visibili.
Il disegno digitalizzato del pesce nel programma dimostrativo si presenta come segue.

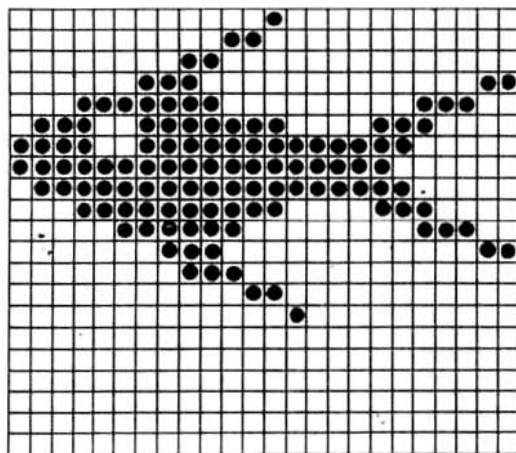



Figura 26.2

Quando il disegno digitalizzato è pronto, occorre convertirlo in istruzioni DATA che possono essere inserite in un programma BASIC. Occorre dividere i pixel che costituiscono l'immagine in byte da 8 bit e quindi convertire ciascun byte nel corrispondente valore decimale. Ecco come procedere:
Prendere la prima riga dell'immagine e suddividerla in tre gruppi di 8 bit come segue:

ØØØØØØØØ ØØØØ1ØØØ ØØØØØØØØ

(Usare un "1" per rappresentare un pixel pieno e uno "Ø" per rappresentare un pixel vuoto).
Convertire ora questi tre numeri binari in decimali, in modo da ottenere

Ø 8 Ø

(Se necessario, vedere pag. 233 per ripassare il metodo).
Ora procedere analogamente per tutte le altre righe dell'immagine.
La seconda riga darà (ØØØØØØØØ, ØØ11ØØØØ, ØØØØØØØØ) che danno luogo rispettivamente a Ø, 48, Ø). La sesta riga (prendendone una a caso) darà (Ø111ØØ11, 11111ØØØ, Ø111ØØØØ) che convertiti daranno a loro volta rispettivamente (115, 248, 112). Si terminerà con 63 numeri decimali che è possibile inserire in istruzioni DATA tipo quelle nelle righe da 3Ø e 9Ø del programma dimostrativo. Occorre fare in modo di introdurre nell'ordine corretto; da sinistra a destra e dall'alto verso il basso.

Naturalmente il processo ora spiegato è insopportabilmente noioso specialmente se si devono disegnare molti Sprites diversi e non si ha troppa dimestichezza con la conversione di numeri binari nei rispettivi equivalenti decimali. In effetti, molte persone non sono abituate a farlo. La vita viene resa molto più semplice usando un adatto strumento di software per disegnare: uno Sprite Editor Program.

ESPERIMENTO 26·2

Caricare ed eseguire il programma MONSPR, che è uno "Sprite Editor". Esso consente di disegnare uno Sprite sullo schermo, spostarlo e ruotarlo o visualizzarlo in "negativo". Se in qualsiasi momento non si sa come procedere, battere "H" in modo da far visualizzare dal programma una spiegazione di tutte le sue funzioni. Infine, quando si è terminata l'immagine dello Sprite, battere "B" per far sì che l'Editor esegua le conversioni da binario a decimale, visualizzando una serie di istruzioni DATA sullo schermo in modo da poterle copiare nel programma.
Una volta prese le misure dello Sprite Editor, usarlo per disegnare il proprio Sprite. Prendere nota delle istruzioni DATA che l'Editor produce e usarle per modificare lo Sprite che è stato appositamente creato invece del pesce.

Esperimento 26.2 Completato	☐

INSERIMENTO DI SPRITES IN UN PROGRAMMA

Si discuterà ora l'inserimento degli Sprites nei programmi. All'interno del programma, la descrizione di uno Sprite richiede 63 byte consecutivi in memoria. Ciascun byte corrisponde a 8 pixel e questi byte sono memorizzati da sinistra verso destra e dall'alto verso il basso nello stesso ordine in cui sono stati calcolati dall'immagine digitalizzata.
Se fosse possibile inserire la descrizione di uno Sprite in 63 celle qualsiasi di memoria adiacenti, si potrebbe semplicemente memorizzarla in una lunga variabile stringa. Sfortunatamente ciò non è possibile.
Per taluni motivi tecnici una definizione di Sprite può soltanto iniziare ad un indirizzo che è un multiplo esatto di 64. Inoltre, la definizione non si può inserire molto facilmente ovunque nell'area compresa fra 4Ø96 e 8191 o oltre 16383.

Se si esamina la mappa di memoria del COMMODORE **64** nell'Unità 23, si vedrà che le aree in cui sono ammesse le definizioni degli Sprite sono in gran parte impegnate dal Kernal, la mappa dello schermo video e dai programmi dell'utente. Si dovrà pertanto fare spazio per gli Sprites spostando qualcosa d'altro.
Se il programma usa soltanto uno o due Sprites, ciò è abbastanza facile. Per combinazione, il blocco delle 63 celle di memoria che inizia a 832 è dedicata ai trasferimenti del nastro su cassetta. Fintantoche non si ha intenzione di effettuare trasferimenti su nastro mentre è in esecuzione il programma Sprite, si possono prendere a prestito queste locazioni per le descrizioni degli Sprites. 832 corrisponde esattamente a 13 volte 64, cosicché è un punto adatto per cominciare. Se si esaminano le righe da 12Ø a 14Ø del programma dimostrativo si vedrà che essi leggono i dati di descrizione dello Sprite e che li inseriscono (POKE) nelle locazioni da 83 in su, un byte alla volta.
Come dire al COMMODORE **64** dove inserire la descrizione dello Sprite?
Potrebbe non sempre essere 832. Per questo lavoro il sistema usa otto celle dedicate all'estremità della mappa dello schermo: quelle con gli indirizzi da 2Ø4Ø e 2Ø47. Si ricorderà che la mappa dello schermo video ha 1Ø24 locazioni, ma di queste ne occorrono soltanto effettivamente 1ØØØ per i caratteri da visualizzare sullo schermo. Questo è il motivo per cui le celle da 2Ø4Ø a 2Ø47 sono effettivamente disponibili per la gestione degli Sprites.
In qualsiasi momento la macchina può visualizzare un massimo di otto Sprites diversi, numerati da Ø a 7. La locazione 2Ø4Ø è sempre collegata allo Sprite Ø; la locazione 2Ø41 allo Sprite numero 1 e così via.
Per qualsiasi Sprite in uso, la cella nella mappa dello schermo contiene un *puntatore* alla corrispondente descrizione dello Sprite. Il puntatore è semplicemente l'indirizzo del primo byte della descrizione *diviso per 64*.
Ad esempio, se la definizione dello Sprite Ø inizia all'indirizzo 832, occorre fare in modo che la cella 2Ø4Ø contenga il numero 13 (poiché 932 diviso per 64 dà esattamente 13). Osservare la riga 3ØØ del programma dimostrativo che esegue esattamente questo lavoro. Essa dice

```
300POKE 2040,13
```

Come verrebbe scritta l'istruzione se il programma dovesse usare lo Sprite numero 4, con la descrizione all'indirizzo 2112? Capovolgere questa pagina per osservare la risposta.

Risposta: 3ØØPOKE 2Ø44,33

IL CONTROLLO DEGLI SPRITES

Gli Sprites in un programma sono visualizzati dallo speciale chip Video Interface Controller (II), un dispositivo estremamente potente ma anche complicato. Se tutti i suoi controlli fossero portati all'esterno su un pannello, ci sarebbero qualcosa come 16 cursori, 3Ø interruttori multipolari, 2Ø spie e un paio di quadranti - sufficienti a riempire la cabina di pilotaggio di un aereo di linea.

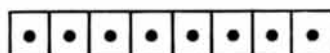
Anche l'unità di controllo video occupa un certo numero di indirizzi consecutivi. La prima di queste locazioni, detta "indirizzo di base" si trova in 53248 o per ricordare meglio, nella forma "2Ø8★256".

Il programma Sprite Demonstration usa una variabile - V - come indirizzo di base dell'unità di controllo. Ciò significa che è possibile fare riferimento alla locazione 21 *dell'unità di controllo* con "V+21" anziché "53269". Ciò rende più leggibile il programma ed evita errori.

In Fig. 26.3 compare una mappa completa dell'unità di controllo video che può occuparsi di un massimo di 8 Sprites alla volta. Si vedrà che alcune delle locazioni sono ombreggiate. Ciascuna di queste appartiene ad un *singolo* Sprite, per esempio la locazione 7 è sempre usata per controllare la posizione verticale dello Sprite numero 3. Altre (quelle non ombreggiate) sono condivise tra tutti gli Sprites, a ciascuno dei quali è allocato un byte degli 8 della locazione. Un esempio è rappresentato dalla locazione 23, che controlla l'espansione verticale di tutti gli 8 Sprites (o di quelli che si trovano correntemente sullo schermo).

Alcune delle locazioni nel chip non hanno nulla a che fare con gli Sprites. Per esempio, la locazione 32 (colore del bordo) e 33 (colore dello sfondo).

Se tutti gli Sprites usano una locazione, i bit sono così disposti:

	•	•	•	•	•	•	•	•
N. Sprite	7	6	5	4	3	2	1	Ø

I registri nell'unità di controllo video sono normalmente letti e modificati dai comandi PEEK e POKE. Si supponga di usare soltanto lo Sprite n. 6 e di espanderlo verticalmente. Per far ciò, occorre inserire il profilo *binario* nella locazione V+23 col comando:

```
POKE V+23,64
```

poiché 64 è l'equivalente decimale di .

Ciò presuppone naturalmente che V sia stato già impostato all'indirizzo di base dell'unità di controllo.

I numeri decimali corrispondenti agli 8 Sprites sono:

N. Sprite	7	6	5	4	3	2	1	Ø
Equiv. decim. inserimento in locazioni condivise	128	64	32	16	8	4	2	1

L'aspetto di uno Sprite sullo schermo dipende da molte cose. Innanzitutto deve essere attivato per essere visibile. Secondo, ha una posizione (può essere cioè definito mediante coordinate), un colore, dimensioni orizzontali e verticali e una priorità che lo inserisce davanti o dietro altri elementi visualizzati sullo schermo.

Gli esempi presuppongono che si stia visualizzando uno Sprite alla volta.

La locazione 21 dell'unità di controllo è usata per attivare e disattivare tutti gli Sprites. A ciascuno è allocato un bit, che compare soltanto se è impostato a valore "1". Per attivare lo Sprite Ø, occorre battere

```
POKE V+21,1
```


come nella riga 33Ø del programma dimostrativo. Notare che "1" è il codice decimale per lo Sprite Ø. POKE V+21,Ø disattiverebbe tutti gli Sprite (compreso lo Sprite Ø).

La posizione dello Sprite (la posizione dell'angolo superiore sinistro della griglia che lo contiene) è definita da due coordinate: X (lunghezza) e Y (profondità).

Il valore di Y può variare tra Ø e 255. Per controllare l'altezza dello Sprite Ø basta inserire (POKE) il valore Y nella locazione n. 1. Per esempio, per inserire lo Sprite Ø 1ØØ unità più in basso, occorre scrivere

```
POKE V+1,1ØØ
```

Ogni Sprite ha un proprio registro Y cosicché per posizionare lo Sprite n. 2 15Ø unità in basso, occorrerebbe scrivere

```
POKEV+5,15Ø
```

Notare che l'allocazione della locazione 5 alla posizione verticale dello Sprite 2 è chiaramente indicata sulla mappa dell'unità di controllo Fig. 26.3

La posizione X di uno Sprite può assumenre qualsiasi valore compreso tra Ø e 511. Ciò può provocare problemi dato che i numeri binari nel campo da Ø a 511 hanno bisogno di *nove* cifre binarie, una in più di quante ne è possibile inserire (POKE) in una singola locazione.

L'unità di controllo divide perciò il valore X in due parti. Gli 8 bit inferiori sono caricati in un registro dedicato (esattamente come il valore Y) ma la cifra più elevata è inserita nella locazione 16, che è condivisa tra tutti gli Sprites ed ha un bit dedicato a ciascuno.

	Registro	Contenuto
(53248)	V	SPRITE Ø - X - COORDINATA (8 BIT INFERIORI)
	V+1	SPRITE Ø - Y - COORDINATA
	V+2	SPRITE 1 (IDEM)
	V+3	
	V+4	SPRITE 2 (IDEM)
	V+5	
	V+6	SPRITE 3 (IDEM)
	V+7	
	V+8	SPRITE 4 (IDEM)
	V+9	
	V+1Ø	SPRITE 5 (IDEM)
	V+11	
	V+12	SPRITE 6 (IDEM)
	V+13	
	V+14	SPRITE 7 (IDEM)
	V+15	
	V+16	X_7 ← BIT PIÙ SIGNIFICATIVI DELLE COORDINATE X → X_0
	V+17	(VEDERE GUIDA DI RIFERIMENTO)
	V+18	
	V+19	
	V+2Ø	
	V+21	E_7 ← BIT DI ABILITAZIONE SPRITE → E_0
	V+22	(VEDERE GUIDA DI RIFERIMENTO)
	V+23	V_7 ← BIT DI ESPANSIONE SPRITE (VERTICALI) → V_0

Figura 26.3

Registro			
V+24	PUNTATORI DI MEMORIA		
V+25	(VEDERE GUIDA DI RIFERIMENTO)		
V+26			
V+27	P_7	← SPRITE - PRIORITÀ DI DATI →	P_0
V+28	M_7	← BIT MULTICOLORI PER GLI SPRITE →	M_0
V+29	H_7	← BIT DI ESPANSIONE SPRITE (ORIZZONTALE) →	H_0
V+3Ø	S_7	← COLLISIONI SPRITE - SPRITE →	S_0
V+31	D_7	← COLLISIONI SPRITE - DATI →	D_0
V+32	COLORE DEL BORDO		
V+33	COLORE DEL FONDO		
V+34	(VEDERE GUIDA DI RIFERIMENTO)		
V+35			
V+36			
V+37	PRIMO COLORE COMUNE		
V+38	SECONDO COLORE COMUNE		
V+39	COLORE SPRITE Ø		
V+4Ø	COLORE SPRITE 1		
V+41	COLORE SPRITE 2		
V+42	COLORE SPRITE 3		
V+43	COLORE SPRITE 4		
V+44	COLORE SPRITE 5		
V+45	COLORE SPRITE 6		
V+46	COLORE SPRITE 7		

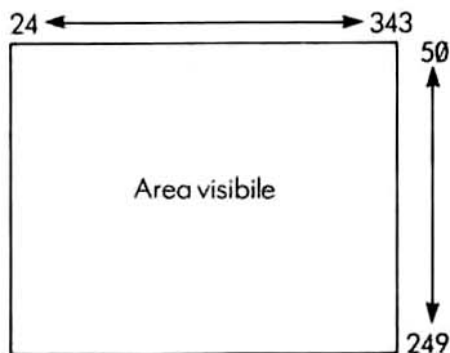
Figura 26.3 (Parte 2)

In BASIC è possibile usare l'operazione AND per separare gli 8 bit inferiori del valore X ed effettuare una prova del tipo "IF X>=256 THEN" per vedere se il bit superiore è un "1". Per inserire lo Sprite Ø nella posizione orizzontale X occorrerebbe scrivere:

```
500 POKE V+Ø , X AND 255
510 IF X>=256 THEN POKE V+16,1:
    GOTO 53Ø
520 POKE V+16,Ø
530 REM X-POSITION NOW SET UP
```

Questi concetti possono essere estesi a qualsiasi Sprite. Per esempio, per inserire lo Sprite numero N nella posizione XN, YN (dove N, XN e YN sono variabili) è possibile scrivere:

```
6ØØ POKE V+2★N , XN AND 255
62Ø IF XN>=256 THEN POKE V+16, 2↑N:
    GOTO 64Ø.
63Ø POKE V+16,Ø
64Ø POKE V+2★N + 1 , YN
```

Questo frammento di programma non funziona però se si sta visualizzando più di uno Sprite! Il motivo viene spiegato più avanti.
Non tutte le posizioni di Sprite sono effettivamente visibili sullo schermo. I confini per l'angolo superiore sinistro di uno Sprite sono indicati sul diagramma che segue:

Questo diagramma mostra che ad esempio, uno Sprite con una coordinata Y di 36 sarebbe soltanto parzialmente visibile mentre uno con una coordinata X di 35Ø non sarebbe visibile per nulla.
Gli altri controlli sono abbastanza semplici. I bit nella locazione 23 si occupano della dimensione verticale degli Sprite e quelli nella locazione 29 controllano la loro dimensione orizzontale. Pertanto il comando

```
POKE V+29,1
```

allungherà lo Sprite numero Ø al doppio della sua larghezza normale. Analogamente

```
POKE V+29,8-PEEK(V+29)
```

cambierà la larghezza dello Sprite numero 3 (la restringerà o la allargherà). È possibile immaginare perché?

I bit nella locazione 27 controllano la priorità degli Sprites: uno "Ø" in qualsiasi bit inserisce il corrispondente Sprite davanti al testo sullo schermo mentre un "1" lo pone dietro di esso.
Infine ciascun Sprite ha una propria locazione dedicata per il controllo del colore. Il registro 39 è pertinente allo Sprite Ø, il registro 4Ø allo Sprite 1 e così via. Per far sì che uno Sprite compaia in un particolare colore, occorre disporre il codice corrispondente (ad esempio 2 per "rosso") nell'appropriato registro del colore.

ESPERIMENTO 26·3

Per dimostrare di aver compreso tutta questa nuova parte, provare praticamente a modificare il Flash Demostration Program in modo che usi lo Sprite numero 7 invece del numero Ø. Il programma dovrebbe funziònare ancora e produrre un'immagine identica sullo schermo. Si troverà utile il diagramma nella Figura 26.3 e non occorrerà cambiare la posizione della descrizione degli Sprite da 832 in avanti.

Esperimento 26.3 Completato	

STUDIO DI QUATTRO PROGRAMMI PER SPRITE

Gli Sprites che rappresentano oggetti in movimento devono essere accuratamente controllati per dare l'impressione di un movimento regolare ed uniforme. Il metodo generale consiste nell'usare un'iterazione che corrisponda ad un intervallo fisso di tempo e per mostrare lo Sprite in una posizione leggermente diversa ad ogni ripetizione. Ciò dà a chi osserva l'illusione del movimento, esattamente come un cartone animato. Per poter risultare efficace, l'intervallo di tempo non dovrebbe superare all'incirca 1/16 di secondo, altrimenti lo Sprite darebbe l'impressione di muoversi a scatti.

Procedendo nell'iterazione, il computer può *calcolare* la posizione successiva dello Sprite oppure cercarla in una tabella. L'iterazione termina quando si verifica qualche *condizione limite* - ad esempio lo Sprite colpisce il bordo dello schermo o un altro oggetto o l'iterazione si è ripetuta per un certo numero prefissato di volte.

Per dimostrare i diversi modi per controllare il movimento dello Sprite, ci sono quattro brevi programmi. Questi non hanno alcun abbellimento tipo effetti sonori o visulizzazioni interessanti a colori; ciascuno è stato ridotto al minimo necessario per illustrare i vari argomenti discussi.

Il tipo più semplice di movimento è quello in linea retta a velocità costante. Caricare il programma LINEAR e quindi dare uno sguardo all'elenco che segue:

```
  5 REM LINEAR
 1Ø REM PROGRAM TO SHOW MOTION
    OF BALL
 2Ø DATAØ,Ø,Ø,Ø,Ø,Ø,112,Ø
 3Ø DATA3,254,Ø,15,255,128,31,255,192
 4Ø DATA63,255,224,63,255,224,127,255,
    240
 5Ø DATA127,255,24Ø,127,255,24Ø,63,255,
    224
 6Ø DATA63,255,224,31,255,192,15,255,128
 7Ø DATA3,254,Ø,Ø,112,Ø,Ø,Ø,Ø
 8Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
1ØØ REM SET UP SPRITE DESCRIPTION
11Ø FOR J=ØTO62
12Ø READA: POKE 832+J,A
13Ø NEXT J
14Ø V=53248 :REM SET BASE ADDRESS
15Ø POKE 2Ø4Ø,13:REM SET POINTER TO
    DESCRIPTION
155 PRINT" SHIFT and CLR HOME ":REM
    CLEAR SCREEN
16Ø POKE V+33,1:REM SET
    BACKGROUND
17Ø POKE V+39,Ø:REM SET COLOUR
18Ø POKE V+23,1:POKEV+29,1:REM
    EXPAND
19Ø X=Ø:Y=Ø:REM SET STARTING
    POSITION
2ØØ DX=2.3 :DY=1.2:REM SET
    HORIZONTAL AND VERTICAL SPEEDS
2Ø5 POKEV+21,1
21Ø POKEV+21,1 :REM ENABLE SPRITE
22Ø REM LOOP STARTS HERE
```

```
230 X=X+DX:Y=Y+DY :REM
    CALCULATE NEW POSITION
240 POKE V+1,Y
250 POKE V+0,X AND 255: REM PUT
    SPRITE THERE
260 IF X >=256 THEN POKE V+16,1:GOTO
    280
270 POKE V+16,0
280 IF Y < 225 AND X < 400 THEN230 :REM
    LOOP AROUND
290 GOTO190 :REM REPEAT MOVEMENT
```

La prima parte del programma imposta una descrizione dello Sprite e inizializza l'unità di controllo video. La parte interessante inizia alla riga 190.
X e Y rappresentano la posizione dello Sprite in qualsiasi momento. Si inizia in (0, 0) che è fuori dallo schermo.
DX e DY sono le quantità di cui X e Y *cambiano* ad ogni intervallo di tempo. Pertanto, dopo un intervallo, Y sarà uguale a 0+1,2=1,2; dopo due intervalli, 2,4; e così via.
Una volta all'interno dell'iterazione, la riga 230 fa avanzare il valore corrente di X e Y. Le righe da 240 a 270 sono responsabili della visualizzazione dello Sprite nella nuova posizione. La condizione limite non viene raggiunta fino a che Y non supera 255 o X non supera 400, quello che viene prima. Mentre la posizione corrente dello Sprite è ancora entro lo schermo, l'iterazione continua imperturbata.

ESPERIMENTO 26•4

Per avere una qualche idea del funzionamento del programma, occorre fermarlo e variare la posizione iniziale e i valori di DX e DY. Si dovrebbe essere in grado di far muovere la palla a diverse velocità e in direzioni diverse. Per far si che la palla si muova verso sinistra o verso l'alto, provare ad usare valori negativi per DX e DY: ma ricordare di alterare la condizione limite e di conseguenza il punto iniziale.

Esperimento 26.4 Completato	

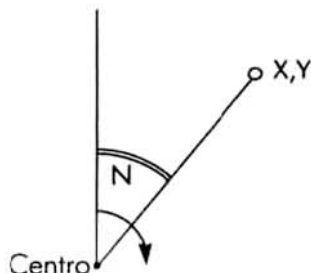
Il programma CIRCULAR1 sposta lo Sprite in un percorso all'incirca circolare. Per conoscere i dettagli del programma, occorre qualche conoscenza di trigonometria elementare. Se ciò crea dei problemi, saltare alla successiva sezione.

Caricare il programma CIRCULAR1 ed eseguirlo. Si vedrà dal listato che segue che il metodo di controllo è abbastanza diverso. Lo Sprite cambia costantemente direzione e non raggiunge mai la fine dello schermo!

```
  5 REM CIRCULAR1
 10 REM PROGRAM TO SHOW MOTION
    OF BALL
 20 DATA0,0,0,0,0,0,112,0
 30 DATA3,254,0,15,255,128,31,255,192
 40 DATA63,255,224,63,255,224,127,255,
    240
 50 DATA127,255,240,127,255,240,63,255,
    224
 60 DATA63,255,224,31,255,192,15,255,
    128
 70 DATA3,254,0,0,112,0,0,0,0
 80 DATA0,0,0,0,0,0,0,0,0
100 REM SET UP SPRITE DESCRIPTION
110 FOR J0TO62
120 READA: POKE 832+J,A
130 NEXTJ
140 V=53248 :REM SET BASE ADDRESS
150 POKE 2040,13:REM SET POINTER TO
    DESCRIPTION
```

155 PRINT " ": REM CLEAR SCREEN

```
160 POKE V+33,1:REM SET
    BACKGROUND
170 POKE V+39,0:REM SET COLOUR
180 POKE V+23,1:POKEV+29,1:REM
    EXPAND
190 FOR N=1 TO 100 STEP 0.15
205 POKEV+21,1
210 POKEV+21,1 :REM ENABLE SPRITE
220 REM LOOP STARTS HERE
230 X=150+ 40★SIN(N):Y=120−30
    ★COS(N)
240 POKE V+1,Y
250 POKE V+0,X AND 255: REM PUT
    SPRITE THERE
260 IF X >=256 THEN POKE V+16,1:GOTO
    280
270 POKE V+16,0
280 NEXTN
290 GOTO190 :REM REPEAT MOVEMENT
```

Si immagini che lo Sprite ruoti intorno alla testa di una persona alla quale è fissato mediante una funicella. La funicella percorrerà lo stesso piccolo *angolo* ad ogni intervallo di tempo. In questo programma, la posizione angolare della funicella è rappresentata da N e la dimensione dell'argomento step è di 0,15 *radianti* circa 7° e mezzo). La posizione della palla dipende dall'angolo N secondo il diagramma che segue:

X=150+40★sin(N)
Y=120-40★cos(N)

Centro del cerchio in (150,120)
Raggio del cerchio=40

In questo caso la "condizione limite" si verifica quando la palla ha compiuto rotazioni pari a 100 radianti - vale a dire circa 16 giri.
Il problema con questo programma è rappresentato dal fatto che è troppo lento per poter dare un senso appropriato del movimento. Ciò in quanto le funzioni SIN e COS usate all'interno dell'iterazione principale di controllo richiedono un lungo tempo di elaborazione.

```
  5 REM CIRCULAR2
 10 REM PROGRAM TO SHOW MOTION
    OF BALL
 20 DATA0,0,0,0,0,0,112,0
 30 DATA3,254,0,15,255,128,31,255,192
 40 DATA63,255,224,63,255,224,127,255,
    240
 50 DATA127,255,240,127,255,240,63,255,
    224
 60 DATA63,255,224,31,255,192,15,255,
    128
 70 DATA3,254,0,0,112,0,0,0,0
 80 DATA0,0,0,0,0,0,0,0,0
100 REM SET UP SPRITE DESCRIPTION
110 FOR J0TO62
120 READA: POKE 832+J,A
130 NEXTJ
140 V=53248 :REM SET BASE ADDRESS
150 POKE 2040,13:REM SET POINTER TO
    DESCRIPTION
```

155 PRINT "   ": REM CLEAR SCREEN

```
160 POKE V+33,1:REM SET
    BACKGROUND
170 POKE V+39,0:REM SET COLOUR
180 POKE V+23,0:POKEV+29,0:REM
    EXPAND
190 DIM XP (80),YP(80)
```

```
200 FOR N=1TO80
210 T= 4.5★N★π/180
220 XP(N)=170+100★SIN(T):YP(N)=130
    —80★COS(T)
230 NEXTN
240 POKEV+21,1 :REM ENABLE SPRITE
250 FOR N=1 TO 80
260 REM LOOP STARTS HERE
270 X=XP(N):Y=YP(N)
280 POKE V+1,Y
290 POKE V+0,X AND 255: REM PUT
    SPRITE THERE
300 IF X >=256 THEN POKE V+16,1:GOTO
    320
310 POKE V+16,0
320 NEXTN
330 GOTO240 :REM REPEAT MOVEMENT
```


CIRCULAR2 illustra un metodo decisamente migliore. Qui sono calcolate in anticipo 80 posizioni lungo il cerchio, memorizzate nelle matrici XP e YP (righe da 190 a 230 del programma precedente), per cui quando lo Sprite è in movimento, la sua successiva posizione può essere ricavata dalla tabella anziché calcolata da capo ogni volta al ripetersi dell'iterazione.
Per il movimento circolare vale sempre la pena di calcolare le posizioni in anticipo e memorizzarle in una tabella. Caricare ed eseguire CIRCULAR2 e notare il miglioramento.
Usando diverse funzioni matematiche per elaborare la tabella, è possibile far sì che lo Sprite si muova su traiettorie interessanti. Ecco alcuni esempi:

```
220 XP(N)=150+100★SIN(3★T):
    YP(N)=120-80★COS(T)
```

o

```
220 XP(N)=150+80★SIN(T)+40★SIN
    (2★T)+20★SIN(3★T)
225 YP(N)=120-60★COS(T)+30★COS
    (2★T)-15★COS(3★T)
```

Una locazione interessante nell'unità di controllo video è quella del registro 31, con un bit per ciascuno degli 8 Sprites. Ogniqualvolta uno Sprite in movimento entra in collisione con un oggetto fisso sullo schermo (non con un altro Sprite ovviamente) il bit che appartiene allo Sprite in movimento viene impostato a valore logico "1". È possibile usare il registro per individuare le palle che rimbalzano contro le pareti, le bombe che colpiscono il terreno, ecc. La tecnica consiste nell'usare il registro come condizione limite leggendolo subito dopo aver spostato uno Sprite. La lettura del registro lo ripristina automanticamente a zero (ma non sempre immediatamente).
Per concludere, si creeranno Sprites che si muovono in caduta libera. Secondo la legge di Newton, la velocità *orizzontale* di un oggetto in caduta libera rimane inalterata ma la velocità verso il basso *aumenta* ogni secondo di una quantità costante della accellerazione. L'accellerazione verso il basso è provocata dalla forza di gravità.

Il nucleo di un programma per imitare un oggetto in caduta libera sarebbe il seguente:

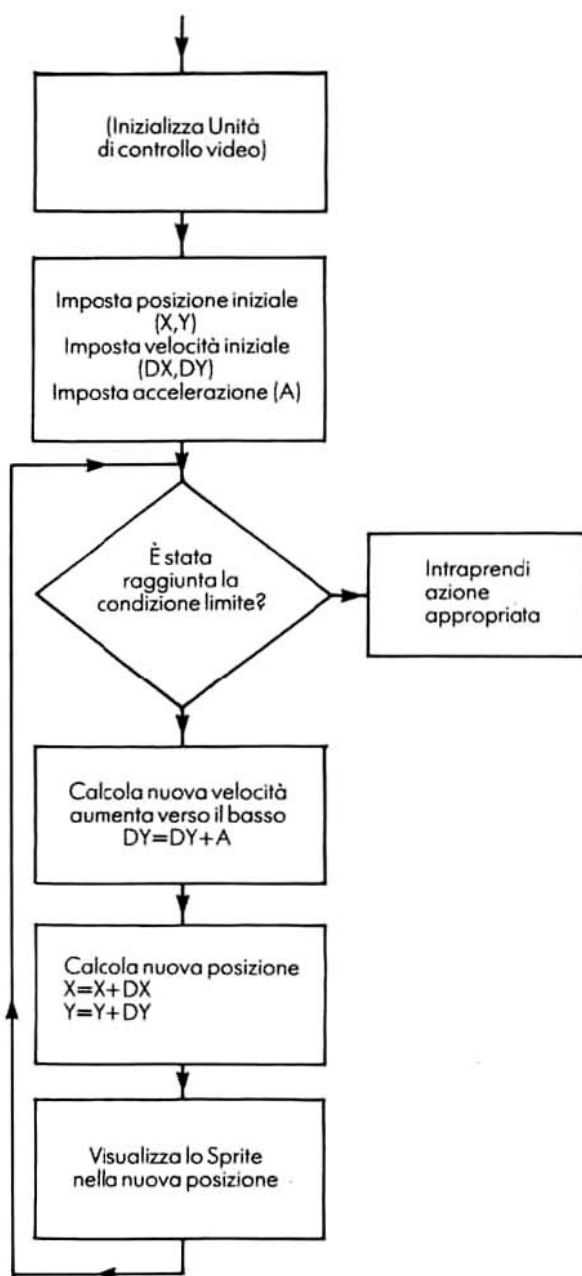
(Inizializza Unità di controllo video)

Imposta posizione iniziale (X,Y)
Imposta velocità iniziale (DX,DY)
Imposta accelerazione (A)

È stata raggiunta la condizione limite? → Intraprendi azione appropriata

Calcola nuova velocità aumenta verso il basso
DY=DY+A

Calcola nuova posizione
X=X+DX
Y=Y+DY

Visualizza lo Sprite nella nuova posizione

```
  5 REM BOUNCING BALL
 10 REM PROGRAM TO SHOW MOTION
    OF BALL
 20 DATA0,0,0,0,0,0,112,0
 30 DATA3,254,0,15,255,128,31,255,192
 40 DATA63,255,224,63,255,224,127,255,
    240
 50 DATA127,255,240,127,255,240,63,255,
    224
 60 DATA63,255,224,31,255,192,15,255,
    128
 70 DATA3,254,0,0,112,0,0,0,0
 80 DATA0,0,0,0,0,0,0,0
100 REM SET UP SPRITE DESCRIPTION
110 FOR J=0TO62
120 READA: POKE 832+J,A
130 NEXTJ
```

```
140 V=53248 :REM SET BASE ADDRESS
150 POKE 2040,13:REM SET POINTER TO
    DESCRIPTION
160 POKE V+33,1:REM SET
    BACKGROUND
170 POKE V+39,0:REM SET COLOUR
180 POKE V+23,0:POKEV+29,0:REM DO
    NOT EXPAND
190 REM DRAW STAIRCASE
200 C$ = " CTRL and ! 1 CTRL
    and RVS ON SPACE 18 times CTRL
    and RVS OFF SPACE 7 times CTRL
    and RVS ON SPACE 19 times"
210 PRINT" SHIFT and CLR HOME CRSR
    12 times"
220 FOR J=1TO13
230 PRINTLEFT$(C$,3★J+2)
240 NEXTJ
250 POKEV+21,1 :REM ENABLE SPRITE
260 X=20:Y=20:REM SET STARTING
    POSITION
270 DX=2.3:DY=0:A=1.4
280 REM LOOP STARTS HERE
285 IFPEEK(V+31)=1ANDDY>0THENDY
    =-0.6★DY:Y=Y+DY:POKEV+1,Y
290 X=X+DX:Y=Y+DY:DY=DY+A
300 POKE V+1,Y
310 POKE V+0,(X AND 255)
320 IF X>=256 THEN POKE V+16,1:GOTO
    340
330 POKE V+16,0
340 IF X < 450 AND Y+DY<255 THEN 285
350 GOTO 260
```

Il principio è illustrato nel programma BOUNCE, che è il caso di caricare ed eseguire. Qui ci sono due condizioni limite. La prima si verifica quando la palla urta un gradino ed è rilevata dalla prova nella riga 285. La condizione (DY>0) assicura che la palla si muove effettivamente verso il basso. Il registro 31 non è talvolta ripristinato immediatamente dopo che è stato letto e se questa condizione viene omessa si può registrare occasionalmente un falso "urto" quando la palla è in movimento.
La "azione appropriata" in questo caso consiste nell'invertire la velocità verticale della palla. Se la palla fosse perfettamente elastica, la velocità verso l'altro dopo aver urtato l'ostacolo sarebbe esattamente la stessa della velocità verso il basso precedente; si potrebbe in tal caso porre "DY=-DY". In pratica, viene conservata soltanto una parte della velocità. "0.6" è all'incirca un valore corretto per una normale palla di gomma. Un rimbalzo di una sfera di acciaio su un gradino di acciaio potrebbe trattenere all'incirca lo 0,9 della sua velocità, mentre una palla floscia ne tratterrebbe soltanto circa la metà.

ESPERIMENTO 26·5

Quando si è studiato il listato del programma, provare a modificare alcuni dei numeri, ad esempio DX (la velocità in avanti), A (l'accellerazione dovuta alla gravità) e la frazione della velocità conservata dopo il rimbalzo. Vedere se è possibile far sì che la palla salti oltre il vuoto nelle scale.

Esperimento 26.5 Completato	

ESPERIMENTO 26·6

Provare a scrivere programmi che usano gli Sprites. Ecco alcuni suggerimenti:

a) un programma che fa muovere uno Sprite in una spirale sempre decrescente;

b) un programma che simula una palla che scorre su un tavolo da bigliardo;

c) un programma che consente di sterzare un bob-sleigh lungo una pista per vedere quale velocità si può raggiungere senza urtare le pareti.

Esprimento 26.6 Completato	

SPRITES MULTICOLORI

Ora verrà illustrata un'altra funzione utilissima: quella che serve per visualizzare gli Sprites in più di un colore.
Come già è noto, uno Sprite ad un colore è costituito da 21 righe di 24 pixel ciascuna. Uno Sprite multicolore è composto a sua volta di 21 righe ma ciascuna riga contiene soltanto 12 pixel. I pixel sono larghi il doppio cosicché il "profilo" completo dello Sprite è lo stesso.
Brevemente, ciascun pixel può essere costituito da uno fra quattro diversi colori:

- Potrebbe essere il colore di *fondo* che è invisibile
- Potrebbe essere un colore particolarmente associato allo Sprite stesso - il cosiddetto colore *primario*
- Potrebbe essere uno dei due colori *comuni* che sono condivisi tra tutti gli Sprites sullo schermo in qualsiasi momento.

I colori effettivi visualizzati dipendono dalla regolazione corrente dell'unità di controllo video e possono essere cambiati di volta in volta.
Per dire all'unità di controllo quale colore usare, ciascun pixel in uno Sprite multicolori usa due cifre binarie. Ecco ciascuna coppia:

ØØ- Colore di fondo
Ø1 - Primo colore comune
1Ø- Colore primario
11 - Secondo colore comune

Come si può vedere, ciascuna linea dello Sprite ha la metà del numero di pixel ma ciascun pixel ha bisogno della doppia quantità di spazio - così la quantità totale di spazio di memoria richiesto per una descrizione di Sprites multicolori è esattamente la stessa di quello per uno Sprites monocolore.
Per dare qualche idea dell'aspetto di uno Sprite multicolori, caricare ed eseguire il programma BUS. È possibile usare questo programma come guida quando è il momento di usare i propri Sprite multicolori.

```
  5 REM GLASGOW BUS
 1Ø REM PROGRAM TO SHOW MOVING
    MULTICOLOUR SPRITE
 2Ø REM DATA DESCRIBING GLASGOW
    BUS
 3Ø DATAØ,Ø,Ø,85,85,84,65,4,4
 4Ø DATA65,4,4,85,85,84,85,17Ø,84
 5Ø DATA255,255,252,195,12,48,195,12,48
 6Ø DATA255,255,255,255,255,255,251,
    255,251
 7Ø DATA8,Ø,8,Ø,Ø,Ø,Ø,Ø,Ø
 8Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
 9Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
1ØØ REM SET UP SPRITE DESCRIPTION
11Ø FOR J=ØTO62
12Ø READA: POKE 832+J,A
13Ø NEXTJ
14Ø V=53248 :REM SET BASE ADDRESS
```

```
150 POKE 2040,13:REM SET POINTER TO
    DESCRIPTION
160 POKE V+1,160 :REM SET
    (CONSTANT) HEIGHT OF SPRITE
170 POKE V+28,1:REM SET MULTI-
    COLOUR
180 POKE V+33,1:REM SET
    BACKGROUND
190 POKE V+37,7:REM SET FIRST
    COMMON COLOUR TO ORANGE
200 POKE V+38,5:REM SET SECOND
    COMMON COLOUR
210 POKE V+39,0:REM SET PRIMARY
    COLOUR
220 POKE V+23,1:POKEV+29,1:REM
    EXPAND
```

230 PRINT" and

16 times C= and P 40 times".

```
240 POKE V,0:POKE V+16,0:POKEV+21,1
250 FOR X=0TO360
260 POKE V+0,XAND255
270 IF X>=256 THEN POKE V+16,1:GOTO
    290
280 POKE V+16,0
290 NEXT X
300 POKE V+21,0
310 GOTO 240
```

Ci sono alcuni punti che occorre ricordare usando questo programma:

1) La versione dello Sprite Editor da usare è detta COSPRED.

2) Per indicare che un dato Sprite usa più di un colore, occorre impostare il bit corretto nel registro 28 dell'unità di controllo.

3) I codici per il colore comune dovrebbero essere caricati nelle locazioni 37 e 38.

SCHERMI CON MOLTI SPRITES

L'ultimo argomento in questa Unità riguarda il problema di visualizzare più di uno Sprite alla volta. Ci sono parecchie grosse difficoltà ma ci sono anche buoni modi per superarle. L'intera sezione suppone che sia stata impostata la varibile V dell'indirizzo di base dell'unità di controllo video è cioè a 208★246.

Il primo problema sta nel cambiare i singoli bit nei registri di controllo dello Sprite senza interessare gli altri nella stessa locazione. Si supponga di voler usare due Sprites - ad esempio 0 e 1. Ad un certo punto del programma si vuole che lo Sprite numero 1 scompaia. Se si impartisce il comando

POKE V+21,0

entrambi gli Sprites scompaiono - non solo quello desiderato. Provare invece

POKE V+21,1

Questa volta il comando funziona (cioè lascia lo Sprite 0 abilitato) ma si sarebbe dovuto sapere *al momento della compilazione del programma* che lo Sprite 0 sarebbe dovuto essere presente in quel momento. In molti casi gli Sprites sono controllati dagli utenti e il programmatore non sa in anticipo quali sono gli Sprites presenti sullo schermo in ogni momento.

Ne deriva che il solo metodo affidabile consiste nell'esaminare (PEEK) il registro di controllo, cambiare il bit desiderato e reinserirlo (POKE). Per cambiare un bit in "1" occorre eseguire un'operazione OR sul byte di controllo con un numero che ha un bit al posto giusto. Per cambiare un bit in uno 0, occorre invece eseguire un'operazione AND (somma logica) con un numero dato dalla tabella che segue:

Numero Sprite	Parola da abilitare OR (imposta 1)	Parola da disabilitare AND (imposta 0)
0	1	254
1	2	253
2	4	251
3	8	247
4	16	239
5	32	223
6	64	191
7	128	127

Ecco alcuni esempi. Per abilitare lo Sprite numero 1 (senza cambiare lo stato dello Sprite 0) occorrerebbe inserire

POKE V+21, PEEK(V+21) OR 2

Per disabilitare lo Sprite 5 (supponendo che lo si stesse usando) il comando corretto sarebbe

POKE V+21, PEEK(V+21) AND 223

Se si sta usando più di uno Sprite, lo stesso metodo deve essere usato con tutti i registri di controllo che contengono bit in comune ad esempio i registri di espansione e di collisione degli Sprites. In particolare, questo è il solo modo sicuro per modificare i contenuti del registro 16 che contiene le cifre più significative delle coordinate X di tutti gli Sprites. Si supponga di voler spostare lo Sprite numero 6 ad una posizione orizzontale di H senza interessare le posizioni degli altri Sprite. Occorre il comando

POKE V+16, (PEEK(V+16) AND 191) OR (INT(H/256)★64)

(Notare che l'espressione

INT(H/256)★64

vale 64 se H è 256 o più, ma vale 0 se H è minore di 256).

Il secondo problema consiste nel trovare un pun-

to adatto per le definizioni degli Sprites. Si ricorderà che ciascuna delle definizioni è lunga 64 byte e che deve iniziare su un confine di "blocco" - cioè in un indirizzo che un multiplo esatto di 64. Sfortunatamente l'area da 832 in su ha spazio soltanto per tre definizioni e se si cerca di memorizzarne di più si va a finire di memorizzare di più si va a finire nell'area dello schermo in 1024.
Ci sono due modi per aggirare questo problema. Uno consiste nell'inserire le definizioni degli Sprite nella parte alta della memoria (ad esempio da 15360 in su) e sperare che il programma e i dati non arrivino a sfiorarli. In caso contrario il programma si guasta inesorabilmente. Il metodo funziona solo per i piccoli programmi.
Un altro sistema molto più sicuro consiste nell'usare la funzione "cambio blocco" del chip dell'unità di controllo. Normalmente il chip vincolato a richiamare tutti i suoi dati - descrizione di Sprite e così via - dal primo blocco di 16K della RAM nella memoria del COMMODORE **64**. Con un cambio di blocco, le definizioni degli Sprite possono ora essere memorizzate nella RAM nascosta "dietro" la ROM del Kernel, come è stato spiegato nell'Unità 24. Sfortunatamente il cambio di blocco ha l'effetto collaterale indesiderato di spostare anche la mappa dello schermo, cosicché i caratteri "ordinari" non possono più essere visualizzati insieme agli Sprites. Il sistema del cambio di blocco di per sè è complicato ma le istruzioni che occorre scrivere sono abbastanza semplici e lineari. Eccole:

1. Memorizzare le definizioni degli Sprites negli indirizzi da 58368 in su. È possibile inserirvele senza dover selezionare la RAM, dato che la ROM del Kernel è "trasparente" quando si memorizzano informazioni negli indirizzi che essa condivide con la RAM sottostante. Si avrà spazio per 112 diverse descrizioni di Sprite - sufficienti per qualsiasi programma -.
Se il programma usa - ad esempio - 12 definizioni di Sprite che sono fornite come istruzioni DATA, le istruzioni per impostarle sono:

```
FOR J=0 TO 12*64-1:READ X:POKE
J+58368,X:NEXT J
```

Ciò presuppone che le ISTRUZIONI per ciascun sprite contengano 64 numeri - 63 per descrivere lo Sprite stesso e uno zero per completare la lunghezza a 64 byte.

2. I "numeri di blocco" di questi Sprite sono "144" per quello che inizia in 58368, "145" per quello che inizia in 58432 e così via, fino a 255 per quello che inizia in 65472. Il cambio di blocco sposterà la mappa dello schermo agli indirizzi da 57374 a 58367. Prima di visualizzare qualsiasi Sprite occorre azzerare la nuova mappa dello schermo e inserire qualche puntatore di blocco degli Sprite negli ultimi otto indirizzi. Per esempio, se si vogliono usare gli otto Sprites con numeri di blocco da 136 a 143 occorrerebbe scrivere

```
FOR J=0TO 999:POKE 57344+J,0:
NEXT J
FOR J=0TO 7:POKE 58360+J,144+J:
NEXT J
```

Naturalmente è possibile ridefinire uno qualsiasi degli Sprite in qualsiasi momento. Se per esempio si desidera che lo Sprite 1 debba essere quello nel blocco 159, occorrerebbe scrivere

```
POKE 57361,159
```

Quando tutte le definizioni degli Sprite e la nuova mappa dello schermo sono state impostate, occorre attivarle mediante i comandi

```
POKE V+24,PEEK(V+24) OR 136
POKE 56578,PEEK(56578) OR 3
POKE 56576,PEEK(56576) AND 252
```

Gli Sprites (un massimo di 8 alla volta) appaiono non appena li si abilita e si inseriscono nelle posizioni corrette nei registri X e Y e ciascuno. Per contro, il normale sistema di schermo del COMMODORE **64** è ora disabilitato e non è possibile PRINT o POKE (rispettivamente scrivere o inserire) caratteri sullo schermo nel modo solito. Non appariranno neppure i messaggi di errore! Se si ritiene che ciò sia essenziale per il programma o se si vuole scoprire esattamente come funziona questo sistema, si consiglia di leggere il manuale di riferimento del programmatore COMMODORE **64**, che contiene una descrizione completa del meccanismo.
A questo punto è utile un'illustrazione. Caricare ed eseguire il programma SOLAR, per il quale viene dato qui di seguito un listato:

```
10 REM SOLAR
20 E=380 :L=27:REM SET NUMBER OF
   ORBITAL POINTS
30 ER=90:LR=25 :REM SET ORBIT SIZES
   OF EARTH AND MOON
40 DIM P2(E),Q2(E),P3(L),Q3(L)
50 PRINT" [SHIFT] and [CLR HOME]
   [CRSR ↑↓] [CRSR ↑↓] [CRSR ↑↓] [CRSR ⇐⇒] [CRSR ⇐⇒] [CRSR ⇐⇒]
   CALCULATING ORBITS PLEASE WAIT"
60 PRINT:PRINT" ABOUT 45 SECONDS!"
240 REM SET UP SPRITE DEFINITIONS
250 FORJ=0TO6*64-1
260 READ X:POKE 58368+J,X
270 NEXT J
290 V=208*256
300 POKE V+32,0:POKE V+33,0 :REM
    SET SCREEN BLACK
320 FOR J=57344 TO 58343:POKE
    J,0:NEXT J:REM CLEAR NEW SCREEN
    MAP
325 FOR J=0 TO 5:POKE58360+J,144+J:
    NEXT J:REM SET SPRITE POINTERS
400 POKE V+41,7:POKE V+42,7:POKE
    V+43,7:POKE V+44,7 :REM SET SUN
    YELLOW
410 POKE V+40,5:POKE V+39,6:REM SET
    EARTH GREEN AND MOON BLUE
420 POKE V+4,160:POKE V+5,130:REM
    POKE COORDINATES OF SUN (4
    QUADRANTS)
430 POKE V+6,184:POKE V+7,130
440 POKE V+8 ,160:POKE V+9 ,149
450 POKE V+10,184:POKEV+11,149
```

```
46Ø PRINT" SHIFT and CLR HOME "
47Ø REM CALCULATE ORBITS
52Ø S=2★π/E: REM NEXT EARTH
53Ø FOR J=1 TO E
54Ø P2(J)=INT(1.2★ER★SIN(S★J)+184):
     Q2(J)=INT(ER★COS(S★J)+149)
55Ø NEXT J
56Ø S=2★π/L: REM NEXT THE MOON
57Ø FOR J=1 TO L
58Ø P3(J)=INT(1.2★LR★SIN(S★J)):
     Q3(J)=INT(LR★COS(S★J))
59Ø NEXT J
68Ø POKEV+24,136 :REM MOVE SCREEN
     AND CHARACTER TABLE
69Ø POKE 56578,PEEK(56578)OR3:POKE
     56576,PEEK(56576) AND 252 :REM
     SELECT BANK 3
7ØØ EM=1:LM=1
7Ø5 POKE V+21,63 :REM ENABLE SPRITES
71Ø FOR Q=1 TO 1ØØØØØ
74Ø POKEV+2,P2(EM) AND 255:POKE
     V+3,Q2(EM)
741 POKE V+16,(PEEK(V+16)AND253)+
     INT(P2(EM)/256)★2
745 PP=P2(EM)+P3(LM):QQ=Q2(EM)
     +Q3(LM)
75Ø POKEV+Ø,PP AND 255:POKE
     V+1,QQ AND 255
751 POKE V+16,(PEEK(V+16)AND254)+
     INT(PP/256)
76Ø EM=EM+1:IFEM>ETHEN EM=1
77Ø LM=LM+1:IF LM>L THEN LM=1
8ØØ NEXT Q
14ØØ REM SPRITE DEFINITIONS
145Ø REM MIDDLE-SIZED PLANET
146Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
147Ø DATAØ,124,Ø,1,255,Ø,3,255,128,3,255,
     128,7,255,192,7,255,192,3,255,128
148Ø DATA1,255,Ø,Ø,124,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,
     Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
15ØØ REM BIG PLANET
151Ø DATAØ,Ø,Ø,24,Ø,1,255,128,7,255,224,
     31,255,248,63,255,252
152Ø DATA127,255,254,127,255,254,255,
     255,255,255,255,255,255,255,255,255,
     255,255
153Ø DATA127,255,254,127,255,254,63,
     255,252,31,255,248,7,255,224,1,255,
     128
154Ø DATAØ,24,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
155Ø REM SUN (1ST QUADRANT)
156Ø DATAØ,Ø,3,Ø,Ø,127,Ø,3,255,Ø,31,255,
     Ø,127,255,1,255,255,3,255,255,7,255,
     255
157Ø DATA15,255,255,31,255,255,31,255,
     255,63,255,255,63,255,255,127,255,
     255
158Ø DATA127,255,255,255,255,255,255,
     255,255,255,255,255,255,255,255
159Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø
16ØØ REM SUN (2ND QUADRANT)
161Ø DATA192,Ø,Ø,254,Ø,Ø,255,192,Ø,255,
     248,Ø,255,254,Ø,255,255,128,255,255,
     192
```

```
162Ø DATA255,255,224,255,255,24Ø,255,
     255,248,255,255,248,255,255,252,
     255,255,252
163Ø DATA255,255,254,255,255,254,255,
     255,255,255,255,255,255,255,255,
     255,255,255
164Ø DATAØ,Ø,Ø,Ø,Ø,Ø,Ø
165Ø REM SUN (3RD QUADRANT)
166Ø DATA255,255,255,255,255,255,255,
     255,255,255,255,255
167Ø DATA127,255,255,127,255,255,63,
     255,255,63,255,255,31,255,255,31,
     255,255
168Ø DATA15,255,255,7,255,255,3,255,
     255,1,255,255,Ø,127,255,Ø,31,255,
     Ø,3,255
169Ø DATAØ,Ø,127,Ø,Ø,3,Ø,Ø,Ø,Ø,Ø,Ø,Ø
17ØØ REM SUN (4TH QUADRANT)
171Ø DATA255,255,255,255,255,255,255,
     255,255,255,255,255
172Ø DATA255,255,254,255,255,254,255,
     255,252,255,255,252,255,255,248,
     255,255,248
173Ø DATA255,255,24Ø,255,255,224,255,
     255,192,255,255,128,255,254,Ø,255,
     248,Ø
174Ø DATA255,192,Ø,254,Ø,Ø,192,Ø,Ø,Ø,Ø,Ø,
     Ø,Ø,Ø,Ø
```

SOLAR usa 6 Sprites: uno per la Terra, uno per la Luna e quattro per i quattro quadrati del Sole, che è troppo grande per essere rappresentato da un solo Sprite. Le definizioni degli Sprites vengono lette dalle istruzioni DATA e memorizzate dal codice nelle righe da 24Ø a 27Ø. La nuova mappa dello schermo viene azzerata e impostata con i puntatori degli Sprites nelle righe 32Ø e 325. Le righe da 47Ø a 59Ø elaborano le tabelle delle orbite della Terra e della Luna cosicché il programma non dovrà eseguire ripetutamente calcoli complessi visualizzando l'immagine in movimento.
Il nuovo banco di memoria viene selezionato dal comando 69Ø e la nuova mappa dello schermo viene impostata dalla riga 68Ø. Infine, la visualizzazione del planetario viene eseguita dalle righe da 71Ø a 8ØØ. Le entrate complesse nelle righe 741 e 751 servono per modificare i singoli bit nel registro 16. Il terzo problema nel visualizzare gli Sprites è la velocità. In SOLAR, il programma appena osservato, i corpi celesti si muovono rapidamente e uniformemente quanto basta per dare una ragionevole impressione di movimento continuo. Se ci sono più di due Sprites in movimento e si tenta di controllarli con un programma BASIC, la visualizzazione rallenta drasticamente; gli Sprites si muovono goffamente e il programma delude anziché divertire. Per rendersi conto, caricare ed eseguire il programma GROTTY, in cui il numero degli oggetti in movimento è stato portato da 2 a 4.
Il modo migliore per far muovere velocemente gli Sprites consiste nell'usare una subroutine scritta in codice macchina, il linguaggio proprio del computer che, dato che non va convertito, viene eseguito molto più velocemente del BASIC.
Il codice macchina è complesso e difficile da ca-

pire e occorrerebbe un intero libro per spiegarlo a fondo; pertanto si fornirà soltanto una adatta subroutine e le istruzioni che occorrono per usarla.

```
 10 DIM SP(1Ø)
1ØØ REM MACHINE CODE ROUTINE TO
    SERVICE SPRITES
11Ø REM IT UPDATES THEIR POSITION
    EVERY INTERRUPT
12Ø DATA165,47,133,251,165,48,133,252,
    16Ø,Ø,177,251,2Ø1,83,2Ø8,8,2ØØ,177,
    251
13Ø DATA2Ø1,8Ø,24Ø,38,136,2ØØ,2ØØ,177,
    251,133,253,2ØØ,157,251,133,254,24,
    165,251
14Ø DATA1Ø1,253,133,251,165,252,1Ø1,
    254,133,252,197,5Ø,2Ø8,212,165,251
15Ø DATA197,49,2Ø8,2Ø6,76,49,234
16Ø DATA152,24,1Ø5,6,168,162,Ø,2ØØ,2ØØ,
    24,177,251,41,1,24Ø,1,56,1Ø2,255,2ØØ
17Ø DATA177,251,157,Ø,2Ø8,2ØØ,177,251,
    157,1,2Ø8,232,232,2ØØ,224,16,2Ø8,225
18Ø DATA165,255,141,16,2Ø8
182 DATA162,Ø,2ØØ,2ØØ,177,251,24Ø,29,
    169,Ø,157,4,212,177,251,157,4,212,169,
    Ø
184 DATA145,251,2ØØ,177,251,157,1,212,
    2ØØ,177,251,157,Ø,212,76,2Ø8,3
186 DATA2ØØ,2ØØ,2ØØ,138,24,1Ø5,7,17Ø,
    224,21,2Ø8,2Ø9,76,49,234
19Ø REM INSERT WEDGE (CODE AT 988)
2ØØ DATA12Ø,169,64,141,2Ø,3,169,3,141,
    21,3,88,96
21Ø FORJ=832 TO 1ØØØ
22Ø READA:POKEJ,A:NEXTJ
23Ø SYS(988):A=2↑31:B=2↑8
```

La routine in codice macchina in SPRMAC si occupa della *posizione* degli Sprites ma non del loro colore o della loro dimensione che cambia rapidamente in un'immagine in movimento, si ha in effetti ciò che si voleva.
Per usare la routine, assicurarsi che sia inclusa nel programma. È possibile listarla e copiarla manualmente ma non lo si consiglia in quanto anche l'errore più piccolo impedirebbe alla routine di funzionare correttamente. È di gran lunga meglio caricare la routine da cassetta o da dischetto e costruire il programma intorno ad essa. La routine dovrebbe trovarsi all'inizio del programma in modo che i primi comandi da eseguire siano quelli nelle righe da 21Ø a 23Ø, che leggono la routine codificata dalle istruzioni dei dati e la accantonano nelle locazioni da 832 a 1ØØØ cosicché quegli indirizzi non possono essere usati per accogliere gli Sprites!
La routine usa le variabili A e B e la matrice SP per scopi speciali. Il programma non deve usare queste variabili salvo come descritto qui di seguito. Una volta che il programma ha eseguito i comandi nelle righe da 21Ø a 23Ø il resto è facile. Per spostare uno Sprite in una data posizione, tutto ciò che si deve fare è di assegnare un singolo valore ad uno degli elementi di SP: SP(Ø) per lo Sprite Ø, SP(1) per lo Sprite 1 e così via.

Il valore dovrebbe essere A+B volte la coordinata X+ la coordinata Y. Per esempio, per collocare lo Sprite 4 nella posizione (31Ø,199) si scrive

```
SP(4)=A+B★31Ø+199
```

È importante che la coordinata X sia un numero intero cosicché per essere sicuri occorre talvolta scrivere

```
SP(2)=A+B★INT(X)+Y
```

Caricare ed eseguire il programma PLANETS, che è indentico a GROTTY salvo che tutti i movimenti degli Sprites sono di pertinenza nella routine in codice macchina. Si vedrà ripristinata l'illusione di movimento uniforme. Le differenze principali tra GROTTY E PLANETS sono le seguenti:

1. PLANETS include la routine in codice macchina in prossimità dell'inizio.

2. I POKE che spostano gli Sprites in GROTTY sono sostituiti da assegnazioni ad elementi di SP. Pertanto il comando che posiziona il primo quarto del Sole viene cambiato da

```
POKE V+6,16Ø:POKE V+7,13Ø
```

in

```
SP(3)=A+★16Ø+13Ø
```

e i comandi che posizionano la cometa sono modificati da

```
POKE V+14,P4(CM) AND 255
POKE V+15,Q4(CM)
POKE V+16,(PEEK(V+16) AND
127)+INT(P4(CM)/256)★128
```

in

```
SP(7)=A+B★P4(CM)+Q4(CM)
```

La stessa routine in codice macchina è anche utile per suonare musica. Ciò sarà illustrato ulteriormente nell'Appendice A.
Questa unità ha fornito soltanto la più breve introduzione possibile all'uso degli Sprites sul COMMODORE **64**. Il chip dell'unità di controllo video ha un potenziale di gran lunga superiore a quello che è stato qui descritto ma occorrerebbe una conoscenza profonda del codice macchina per ottenerne le massime prestazioni. Qualcuno un giorno ci scriverà magari un libro!

CONCLUSIONE

Congratulazioni! Avete raggiunto la fine del corso e se lo avete seguito accuratamente svolgendone tutti gli esempi, avrete avuto un approccio alle problematiche generali del linguaggio BASIC. Avrete iniziato ad apprezzare l'immensa potenza del computer per fornire diletto, divertimento e utile servizio a tutti. Avrete l'esperienza e la conoscenza per applicare la vostra macchina ai giochi, agli affari, alle previsioni, per aiutare l'insegnante in classe e in molti altri campi. Con tutta probabilità preferireste interrompere lo studio e usare la vostra esperienza in vari affascinanti ed eccitanti progetti di cui avete sognato durante il corso.
Il BASIC è sotto molti aspetti un eccellente linguaggio con il quale imparare la programmazione, ma c'è un serio inconveniente che deve essere citato: non è standardizzato. Ciò significa che la versione del BASIC che troverete sulle diverse macchine, è generalmente leggermente diversa e solitamente inferiore al BASIC del **64**
Alcuni BASIC vi danno una scelta notevolmente ristretta di nomi variabili e molti non consentono addirittura l'uso delle stringhe. Le regole sull'inserimento di parecchi comandi su una riga non sono per nulla universali e gli arrangiamenti all'interno dei comandi PEEK, possono anche variare tra le varie macchine. Ci sono inoltre altre piccole differenze troppo numerose da citare: per esempio PEEK e POKE hanno risultati totalmente diversi. Se avete in progetto di trasferire i vostri programmi su qualsiasi altra marca di computer, dovete conoscerne le limitazioni prima di disegnare il programma, altrimenti vi troverete in serie difficoltà.
Terminiamo con 10 "precetti" di buona programmazione. Oscar Wilde una volta ha detto: "Date sempre dei buoni consigli ad altri; è la sola cosa che potete fare".
È in questo spirito che viene offerto questo consiglio: prendetelo o lasciatelo, come preferite:

1) Puntate alla perfezione. Ogni parte del vostro programma e della sua documentazione devono essere il meglio che sapete fare.

2) Progettate prima di costruire. Decidete cosa deve fare il programma prima di pensare a come farlo.

3) Siate preparati a buttare il tutto e ricominciare da capo. Non fate l'errore di innamorarvi di ciò che avete già fatto. Ricordate che ci sono molti modi per risolvere i problemi e il primo metodo che vi viene in mente, difficilmente è il migliore.

4) Pianificate e registrate il vostro schema per allocare le variabili. Il glossario è un documento di lavoro che deve essere usato costantemente quando scrivete i vostri programmi.

5) Dove occorre, scegliete buoni algoritmi. Per esempio, usate una ricerca dicotomica invece di una ricerca "diretta" o Quicksort invece di Bubble sort. Non ha importanza se la lista da esaminare o da riordinare è molto corta — usate sempre il metodo più semplice che potete trovare.

6) Fate attenzione all'interfaccia con l'utente. Dove è appropriato, assicuratevi che i vostri programmi possano essere usati da chiunque senza speciali istruzioni.

7) Usate il lavoro di altri. Non scrivete mai una subroutine se potete trovarne una valida in una libreria di programmi.

8) Non provate nulla di difficile o complicato ma suddividete il lavoro in fasi molto semplici. Usate le subroutine e osservate le convenzioni.

9) Evitate i cosiddetti "trucchi di programmazione intelligente" specialmente se non conoscete come funzionano.

10) Trovate sempre un'altra persona alla quale far compiere la prova finale del vostro programma.

Il nostro viaggio insieme attraverso il BASIC è terminato. Buona fortuna e buona programmazione!

Andrew Colin — Glasgow, 1983

APPENDICI

APPENDICE D	PAGINA 309
APPENDICE E	318
APPENDICE F	326

Il COMMODORE **64** dispone di un sintetizzatore sonoro incorporato che può essere usato come strumento musicale versatile e insolito. Occorre molta attenzione e pianificazione per ottenere le migliori prestazioni della macchina e occorrerà talento artistico e scienza. Quando si esegue la prova finale di un programma musicale non si dice "è corretto" ma "suona bene?" e questo è in definitiva una questione di gusto personale.
Questa Appendice rappresenta soltanto un'introduzione al modo di fare musica sul **64**. Il chip del sintetizzatore sonoro è così flessibile ed ha così numerose funzioni che occorrerebbe in intero libro per descriverlo completamente e spiegarne l'uso per ottenere i migliori effetti. Un giorno forse tale libro verrà anche scritto ma nel frattempo la fonte migliore di ulteriori informazioni è la guida di riferimento del programmatore COMMODORE **64**.
Una parte vitale del far musica consiste nel produrre le singole note. L'argomento è stato discusso nell'Unità 13. Si raccomanda vivamente di rileggere quell'unità e di provare di nuovo il programma ESG prima di continuare con questa Appendice. In particolare, occorre avere familiarità con le nozioni di altezza, durata e timbro e con l'idea dell'inviluppo di una nota.

1. ESECUZIONE DI MOTIVI

Un motivo è una sequenza di note, ciascuna con una propria altezza e durata. Il timbro e l'inviluppo sono solitamente comuni a tutte le note dato che queste verrebbero normalmente suonate sullo stesso strumento.
Si sa già come far suonare alla macchina le note una alla volta. In principio per suonare un motivo, tutto ciò che occorre fare è di fornire al **64** i particolari di ciascuna nota e fargli eseguire uno dopo l'altro. Rimangono da chiarire soltanto alcuni dettagli:
Nell'Unità 13 si è spiegato come variare l'altezza di una nota, inserendo (POKE) numeri negli indirizzi 54272 e 54273. L'altezza dipendeva dalla frequenza della nota ossia dal numero delle vibrazioni al secondo e dalla corretta sequenza di comandi F, era:

```
FF=16★F
POKE 54273,FF/256
POKE 54272,(FF-32768) AND 255
```

Per suonare un motivo, occorre conoscere la frequenza di ogni nota. Sarà di aiuto la seguente tabella:

Successivamente occorre specificare la lunghezza di ciascuna nota. Ciò viene più facilmente fatto in battute, dove la battuta è la lunghezza della nota più corta nel pezzo.
Qui c'è un programma per suonare la prima frase del motivo in Figura A2. Ciascuna istruzione DATA (salvo l'ultima) descrive una nota: i due numeri rappresentano la frequenza e la lunghezza (in crome). Per esempio, la prima nota è una semiminima in Re. La sua frequenza (ricavata dalla Fig. A1) è 287 e la sua durata è di due crome.
Ciò dà:

```
DATA 287,2
```

L'ultima istruzione DATA contrassegna la fine del motivo usando lo speciale valore di frequenza zero.

```
  10 REM IN DUBLIN'S FAIR CITY
  20 VV=212★256 :REM SET BASE
     ADDRESS OF SOUND CHIP
  30 POKE VV+24,15 :REM SET VOLUME
  40 POKE VV+5, 9+16★1 :REM SET
     ATTACK AND DECAY
  50 POKE VV+6,Ø:REM SET RELEASE
     VALUE
  60 REM MAIN LOOP STARTS HERE
  70 READ F,N : REM READ DETAILS OF
     NEXT NOTE
  80 IF F=Ø THEN POKE VV+24,Ø:STOP:
     REM Ø MEANS "END OF TUNE"
  90 FF=16★F : REM SET FREQUENCY OF
     NOTE
 100 POKE VV+1,FF/256
 110 POKE VV,(FF-32768)AND 255
 120 T=TI+15★N :REM SET ALARM FOR
     DURATION
 130 POKE VV+4,17 :REM START NOTE
 140 IF TI < T THEN 140 :REM WAIT FOR
     ALARM
 150 POKE VV+4,Ø :REM STOP NOTE
 160 GOTO 60
 200 DATA 287,2:REM IN
 210 DATA 384.2:REM DUB-
 220 DATA 384,2:REM LIN'S
 230 DATA 384,2:REM FAIR
 240 DATA 384,1:REM CI-
 250 DATA483,3 :REM TY
 260 DATA384,1 :REM WHERE
 270 DATA384,1 :REM THE
 280 DATA431,2 :REM GIRLS
 290 DATA431,2 :REM ARE
 300 DATA431,2 :REM SO
 310 DATA431,1 :REM PRET-
 320 DATA512,3 :REM TY
1000 DATA Ø,Ø, :REM END OF TUNE
```

Il programma inizia impostando la variabile VV all'indirizzo di base del chip sonoro. Questo è a 54272 o in una forma che potrebbe essere più facile da ricordare, 212★256. Tutti i registri nel chip sonoro possono ora essere indicati come spostamenti dall'indirizzo di base invece di usare i loro indirizzi assoluti. Ciò rende il programma più breve e più facile da comprendere nonché riduce la possibilità di errori.

I registri di controllo per la voce che si sta usando sono tutti vicini come segue:

Locazione assoluta	Locazione relativa	Scopo
54272	(VV+)Ø	Controllo altezza
54273	(VV+)1	Controllo altezza
54274	(VV+)2	Controllo ampiezza impulso
54275	(VV+)3	Controllo ampiezza impulso
54276	(VV+)4	Seleziona forma d'onda e inizio/arresto inviluppo
54277	(VV+)5	Controllo attacco e attenuazione
54278	(VV+;6	Controllo tenuta e rilascio

Il registro di controllo di volume è all'indirizzo 54296 oppure (in termini relativi), VV+24.
Quando l'indirizzo di base è stato impostato in VV, il programma regola il volume al massimo e seleziona i valori di attacco e di attenuazione per dare una nota che sfuma dolcemente come un'arpa irlandese. I valori di tenuta e di rilascio non occorrono cosicché il registro VV+6 è impostato a zero. Queste regolazioni rimarranno in vigore per tutta la durata del motivo.
L'iterazione principale del programma, che viene eseguita una volta per ogni nota nel motivo, inizia alla riga 6Ø. La macchina legge i valori di frequenza e di durata per la successiva nota. Se la frequenza è Ø, significa che il motivo è terminato e il programma pertanto interrompe il suono e si ferma. Altrimenti imposta i registri di altezza (nelle righe da 9Ø a 11Ø) e imposta un "allarme" per temporizzare la nota. Ogni battuta è stata impostata a 15 jeffies e la variabile T è impostata al tempo quando la successiva nota deve terminare. Per esempio, se la nota deve durare tre battute, T è impostato al tempo corrente più 3★15 ossia 45 jeffies.
La riga 13Ø inizia la nota e quindi la riga 14Ø fa sì che la macchina attenda fino a che il temporizzatore interno TI non si metta alla pari con il tempo di allarme in T. Quando ciò si verifica, la macchina interrompe il suono (inserendo (POKE) Ø nel registro in VV+4) e ritorna indietro per eseguire la nota successiva.

Si troverà una copia di questo programma su nastro o sul dischetto sotto il titolo DUBLIN. Caricarlo ed eseguirlo. Provare quindi qualche esperimento.
Innanzitutto estendere il programma in modo che suoni l'intero motivo (e non soltanto la prima riga).
Successivamente, cambiare la velocità modificando il "15" alla riga 120 con qualche altro numero; per esempio, "30" dimezzerà la velocità del motivo.
Infine, provare a cambiare il timbro regolando le righe 40, 50 e 130. Per esempio, quanto segue dà un suono tipo tromba:

```
 40 POKEVV+2,200:POKE VV+3,0
    :POKE VV+5,16
 50 POKE VV+6,240
...
130 POKE VV+4,65
```

Esperimento A3.1 completato	

L'esperimento A3.1 avrà convinto che immettere musica nel **64** è un lavoro molto difficile e predisposto agli errori. Un altro inconveniente è che ciascuna istruzione DATA richiede 16 byte e in un programma molto grande non si potrebbe economizzare nè spazi di memoria per un motivo piuttosto lungo.

Altezza

Durata

Figura A3

Per rendere le cose più facili, si userà una notazione speciale che rappresenta ciascuna nota con due caratteri - uno per la sua altezza e uno per la sua durata. Il codice è indicato in figura A.3. È stato inventato specialmente per questo manuale ed è diverso da qualsiasi altro codice, in particolare non è lo stesso codice usato nel 64 SUPER-EXPANDER. Ciononostante ci si abituerà presto a codificare i motivi.
Come si vedrà, le note intervallate da un semitono sono rappresentate da caratteri con codici ASCII adiacenti. Per esempio i codici dei simboli ? @ A B C sono i numeri 63,64,65,66 e 67. Questa configurazione rende facile cercare i valori esatti da inserire (POKE) da una tabella. Una pausa (l'assenza di suono) è indicata dal caratterere "(" (parentesi sinistra).
Le lunghezze delle note sono misurate in quarti di nota o semicrome. Dove una nota dura più di 9 semicrone, ci limitiamo a contare alla rovescia nel codice ASCII cosicché (ad esempio) una "minima puntata" | che è lunga 12 semicrome, è indicata com "<". Analogamente (16 semicrome) è scritta come "@".
Per scrivere un motivo occorre inserire due stringhe. Una stringa contiene tutti i caratteri di altezza, e la seconda tutti i caratteri di durata. Si usano tante coppie di stringhe quante sono necessarie e si fa seguire l'ultima con una "Z".
La versione codificata di "Dublin's Fair City" è la seguente:

```
DATA "CHHHHLHHJJJJMJLJHOMLLJHJ"
DATA "444426224442644444444448"
DATA "CCHHHHHLHJJJJMJJLOMLOMLHJ
      H"
DATA "22444264444262226426444448"
DATA "Z"
```

Qui di seguito è indicato un altro esempio, che è scritto in chiave di basso e usa qualche pausa.

(da "Elijah" di Mendelssohn)

Figura A4

```
DATA "A<(999:<(5<<> AA(,95????(9:<"
DATA "44211314222222222222221146222"
DATA ">>>>>:766(6999>>>CCCB"
DATA "4312222222224314226284"
DATA "Z"
```

Ecco invece un programma che esegue motivi scritti in questa notazione.

```
10 REM TUNE PLAYER
20 DIM N(60) :REM FREQUENCY TABLE
30 FOR J=1 TO 60 :REM SET FREQUENCY
   TABLE
40 N(J) = 64★1.059463 ↑ J
50 NEXT J
55 R=6
60 VV=212★256:REM SET BASE ADDRESS
70 POKE VV+24,15 :REM SET VOLUME
80 POKE VV+5,9+16★1 :REM SET
   ATTACK AND DECAY
90 POKE VV+6,0 :REM SET RELEASE
   VALUE
100 REM MAIN LOOP STARTS HERE
110 READ X$
120 IF X$="Z" THEN POKE VV+24,0:STOP
   :REM STOP IF END OF TUNE
130 READ Y$
140 FOR J=1 TO LEN(X$):REM PLAY EACH
   NOTE IN STRING PAIR
150 A=ASC(MID$(X$,J,1)):B=ASC(MID$
   (Y$,J,1))
160 FF=16★N(A-40)
170 POKE VV+1,FF/256
180 POKE VV,(FF-32768) AND 255
190 T=TI+R★(B-48):REM SET ALARM IN T
200 POKE VV+4,17 :REM START NOTE
210 IF TI<T THEN 210 :REM WAIT FOR
   NOTE TO END
220 POKE VV+4,0 :REM STOP NOTE
230 NEXT J
240 GOTO100
400 DATA"CHHHHLHHJJJJMJLJHOMLL
   JHJ"
410 DATA"444426224442644444444448"
420 DATA"CCHHHHHLHHJJJJMJJLOM
   LOMLHJH"
430 DATA"22444262244426222642646248"
440 DATA"Z"
```

Il programma è fondamentalmente simile alla prima di questa sezione ma ci sono alcune importanti differenze.
Innanzitutto il programma elabora una propria tabella di frequenza. La tabella è conservata in una matrice N e i valori sono elaborati nelle righe da 30 a 50. La formula matematica nella riga 40 è quella che correla l'altezza alla frequenza e le cose sono disposte in modo che la frequenza del Do centrale sia messo in N(25). Ciascun elemento nella matrice corrisponde ad una nota diversa sul pianoforte, cosicché ad esempio N(26) contiene la frequenza di Do diesis (ossia Re bemolle) mentre N(44) contiene il Sol un ottava sopra il Do centrale. Il Do basso è in N(1) e il Do alto in N(49). N(40) contiene 0, una "frequenza" che serve per eseguire una pausa.
Successivamente il programma imposta l'indirizzo di base e i vari registri di controllo in modo diretto. L'iterazione principale inizia con la riga 110 che legge la stringa dell'altezza in X$. Se si trova una "Z" il programma termina dato che "Z" è usato per contrassegnare la fine del motivo.

Altrimenti il programma legge la stringa "durata" in Y$ e immette l'iterazione interna in 14Ø. Questa iterazione viene eseguita una volta per ciascuna nota nelle stringhe. La variabile J è usata per far eseguire entrambe le stringhe contemporaneamente, estraendo ogni volta un'altra copia di caratteri. I caratteri sono convertiti nei rispettivi codici ASCII e inseriti nelle variabili A e B. Quindi viene usato A per indicizzare la tabella di frequenza e B è usato per controllare la lunghezza della nota. Lo spostamento (4Ø) è richiesto nella riga 16Ø in quanto il codice ASCII per "A" (che sta per il Do centrale) è 65, ma la frequenza per il Do centrale è memorizzata in N(25). 65 - 25=4Ø. Lo spostamento 48 nella riga 19Ø è richiesto per un motivo analogo.
Quando viene scelta la lunghezza di una nota, il tempo è moltiplicato per la variabile R. Il valore di R è impostato nella riga 55 e i valori diversi fanno sì che il motivo accelleri o rallenti di conseguenza.
Questo programma può essere facilmente adattato per eseguire qualsiasi motivo a piacere.

2. ARMONIA

Il COMMODORE **64** ha tre "voci" separate e distinte, solo una delle quali è stata finora usata. I registri di controllo sono nelle posizioni (relative) da 7 a 13 sul chip sonoro e quelli per la voce 3 sono nelle posizioni da 14 a 2Ø. Essi vengono usati esattamente allo stesso modo dei controlli della voce 1.
Dato che le tre voci possono venir eseguite contemporaneamente, il **64** può eseguire accordi e musica in tre parti separate. Inoltre, le parti possono avere timbri diversi in modo che la macchina si comporti come un insieme di strumenti diversi.
Sfortunatamente quando si tenta di scrivere un programma che implementa questi concetti, i risultati sono molto scoraggianti; le note non suonano al momento giusto e l'esecuzione ricorda quella di un gruppo di dilettanti poco preparati. Il motivo di ciò va ricercato nella velocità alla quale il BASIC è interpretato dal **64**. Ogni nota ha bisogno di parecchi comandi per poter essere eseguita e dato che questi comandi possono essere avviati uno alla volta, si avverte chiaramente che le note che dovrebbero suonare insieme in realtà partono una dopo l'altra.
Se ci attenesse strettamente al BASIC non ci sarebbe rimedio per questo problema. Il solo modo per risolvere la difficoltà consiste nel fornire al computer parte del programma in codice macchina, il linguaggio che il processore può eseguire direttamente senza conversione. Il codice macchina viene eseguito circa 1ØØØ volte più velocemente del BASIC ma è difficile da scrivere e non è consigliato se non per le applicazioni più critiche (di questo tipo, per esempio).
Il programma fornito qui di seguito comprende una sezione in codice macchina (nelle righe da 1Ø a 1ØØ) che consente di suonare simultaneamente tutte le note in un pezzo in tre parti. È possibile caricare il programma dalla cassetta o dal dischetto (è detto "SHEBA") e provarlo.
Se si conosce la notazione musicale descritta nella precedente sezione, è possibile facilmente adattare il programma per suonare qualsiasi musica a piacere. Occorre cambiare le istruzioni DATA dalla riga 1ØØØ in su inserendo il codice per la propria musica.
Gli aspetti dettagliati di questo programma che si potrebbero modificare sono i seguenti:

a) La velocità di esecuzione è governata dal *secondo* numero nell'istruzione DATA 31Ø. Maggiore è il numero e più lenta è la musica.

b) Il tipo di forma d'onda usata per ciascuna delle tre voci è controllata dal primo numero nelle istruzioni DATA 32Ø, 33Ø e 34Ø rispettivamente. Nel "Arrival of the Queen of Sheba" la voce superiore usa una forma d'onda a impulsi che dà un suono metallico mentre le due voci più basse usano forme d'onda triangolari che hanno caratteristiche più dolci. L'insieme dovrebbe assomigliare ad un clavicembalo a due manuali.

c) Gli altri registri sonori sono impostati nelle righe da 5ØØ a 565 ed è possibile modificare questi comandi per produrre il tipo di suono più gradito.

d) La musica vera e propria è data nelle istruzioni DATA dalla riga 1ØØØ in su. Le istruzioni sono a gruppi di sei, ciscuno dei quali descrive circa quattro battute di musica. Le prime due stringhe danno l'altezza e la durata delle note nella prima voce; le stringhe tre e quattro si riferiscono alla voce 2 e le ultime due stringhe descrivono la voce numero 3. Le prime 8 battute del pezzo sono scritte per esteso in modo da poter studiare la notazione.

```
1 REM QUEEN OF SHEBA
2 REM BY G F HANDEL (ARRANGED
  FOR COMPUTER BY A J T COLIN)
3 POKE 5328Ø,Ø:POKE 53281,1
4 PRINT " [SHIFT] and [CLR HOME] [CRSR ↓]
  3 times [CRSR ⇒] 4 times [CTRL] and
  [1] THE ARRIVAL OF"
5 PRINT " [CTRL] and [1] [CRSR ↓]
  3 times [CRSR ⇒] 4 times [SPACE]
  1Ø times THE"
6 [CTRL] and [1] [CRSR ↓] 3 times
  [CRSR ⇒] 4 times [CTRL] and [3]
  QUEEN OF SHEBA"
7 PRINT " [CTRL] and [1] [CRSR ↓]
  6 times [CRSR ⇒] 3 times BY G F HANDEL"
1Ø REM SHEBA
2Ø DATAA5,Ø1,29,FE,85,Ø1,AD,ØØ,AØ,
   DØ,Ø9,A5,Ø1,Ø9,Ø1,85,Ø1,4C,31,EA,
   A2,ØØ
3Ø DATABD,82,AØ,FØ,6D,FE,87,AØ,DØ,
   68,FE,88,AØ,DØ,63,BD,83,AØ,85,FE,
   BD,84,AØ
```

```
40 DATA85,FF,A0,00,B1,FE,0A,A8,B9,
   02,A0,9D,00,D4
50 DATAB9,03,A0,9D,01,D4,BD,85,A0,
   85,FE,BD,86,A0,85,FF,A0,00,B1,FE,
   D0,06
60 DATA9D,82,A0,4C,C8,03,A8,BD,87,
   A0,38,ED,01,A0,9D,87,A0,BD,88,A0,
   E9,00
70 DATA9D,88,A0,88,D0,EB,FE,83,A0,
   D0,03,FE,84,A0,FE,85,A0,D0,03,FE,
   86,A0,A9,00
80 DATA9D,04,D4,BD,82,A0,9D,04,D4,
   8A,18,69,07,AA,C9,15,D0,03,4C,4B,
   03,4C,56,03
90 REM HERE COMES WEDGE
   INSERTION CODE
100 DATA78,A9,40,8D,14,03,A9,03,8D,
   15,03,58,60
200 FOR J=832 TO 995
210 READ X$:L=ASC(LEFT$(X$,1)):R=ASC
   (RIGHT$(X$,1))
220 IF L>=65 AND L<=70 THEN Y=16★
   (L-55):GOTO 240
230 Y=16★(L-48)
240 IF R>=65 AND R<=70 THEN Y=Y+
   R-55:GOTO 260
250 Y=Y+R-48
260 POKE J,Y
280 NEXT J
300 REM SET UP CONTROL AREA
310 DATA 0, 8
320 DATA 65,151,160,103,168,255,255
330 DATA 17,55,176,7,184,255,255
340 DATA 17,215,191,167,199,255,255
350 READ A,B
360 POKE 40960,A:POKE 40961,B
370 F=1100:Q=2↑(1/12)
375 POKE 40962,0:POKE40963,0
380 FOR J=1TO63
390 POKE 40962+2★J,(F-32768)AND 255
400 POKE 40963+2★J,INT(F/256)
410 F=F★Q
420 NEXTJ
430 FOR J=0 TO 20: READ A: POKE
   41090+J,A:NEXTJ
500 REM SET UP VOICES
510 VV=212★256
520 FOR J=0 TO 24:POKE VV+J,0
530 POKE VV+5,25:POKE VV+12,25:
   POKE VV+19,25
560 POKE VV+24,79:POKE VV+2,200:
   POKE VV+9,60 :POKE VV+16,60
565 POKE VV+22,50 :POKE VV+23,0
570 IFP=0THEN600
580 FORJ=1TOP:READ A$,B$,C$,D$,E$,
   F$:NEXT
600 REM SET UP CONTROL AREAS
610 FOR J=0 TO 2
620 READ X$,Y$
621 IF X$="Z" THEN 750
630 Z=LEN(X$)
640 FOR Q=1 TO Z
650 N=ASC(MID$(X$,Q,1))-40: POKE
   41111+J★4000 + P(J),N
660 N=ASC(MID$(Y$,Q,1))-48: POKE
   43111+J★4000 + P(J),N
670 P(J)=P(J)+1
680 NEXT Q
690 NEXT J
700 POKE 40960,1
705 SYS(983)
710 GOTO 600
750 REM PLANT END OF TUNE
760 FOR J=0 TO 2
770 POKE 41111+J★4000+P(J),0
780 POKE 43111+J+4000+P(J),0
790 NEXT J
800 GOTO800
1000 DATA"WRORWRORWRORWROR
   MWVTRPOMOKMOPRTVWRKRWR
   ORWQMQWTMTVTRTMQRMOMO
   MO"
1010 DATA"1111111111111111111111111
   11111111111111111111111111
   222111111"
1020 DATA"??>><<::88::?377<<5599:
   CE5:25"
1030 DATA"2222222222224422222222222
   2422"
1040 DATA"332200..,,..33++0055--.
   ?A5.25"
1050 DATA"2222222222224422222222222
   2422"
1060 DATA"MJFJMJFJROKOROKOPMH
   MPMJMOFHJKMOPROKOROKOR
   MJMRMJMOKHKOKHKOJFJOJFJ"
1070 DATA"1111111111111111111111111
   11111111111111111111111111111
   11111111"
1080 DATA"::8877<<55::?3CFCFAFAF?C
   ?C>C>C"
1090 DATA"2222222222224422222222222
   22222"
1100 DATA"..,,++00))..33?C?C>A>A<?
   <?:>:>"
1110 DATA"2222222222224422222222222
   22222"
2320 DATA"TRTRPOPOMOMOPRPRTRTR
   POPOMOMOPOPMORPTRVTWVT
   RWPOMKFJK"
2330 DATA"1111111111111111111111111
   11111111111111211224224480"
2340 DATA"PRPRMOMOJOJOMRMR
   PRPRMOMOJOJOMOPMFJKOKH
   FHFD?"
2350 DATA"1111111111111111111111111
   111111122442244480"
2360 DATA"::::::::::::::::3578:878:8::3"
2370 DATA"2222222222222222222222222
   444480"
2400 DATAZ,Z
```

Figura A7

LIBRERIA DI PROGRAMMI

Questa sezione contiene una piccola libreria di utili subroutine. Tutte le subroutine sono state accuratamente controllate. Di regola, esse non variano i valori dei rispettivi parametri di input a meno che le stesse variabili siano anche specificate come parametri di output. (Un'eccezione è la subroutine per risolvere le equazioni simultanee).

Quando si disegna un nuovo programma, occorre scegliere e incorporare una qualsiasi di queste subroutine necessarie; come risultato, il programma sarà più affidabile e funzionerà più rapidamente.

Per usare la libreria, caricare il programma (dal nastro o dal dischetto) e svolgere il menu rispondendo "sì" o "no" a ciascuna subroutine. Dopo l'ultima opzione il programma LIBRARY si cancella da solo e lascia soltanto le subroutine che effettivamente occorrono. A questo punto occorre salvare (SAVE) le routine su un nastro o dischetto separato.

Notare che il programma LIBRARY non può essere riavviato se non ricarecandolo. Se viene interrotto a metà, può lasciare il COMMODORE ***64*** *in una condizione non standard, che non gli consente di caricare i programmi correttamente. Ciò può sempre essere rimediato spegnendo il computer e riaccendendolo.*

Le subroutine sono state disposte in quattro sezioni:

Sezione A: Input da tastiera

1. Input tollerante: accetta I e O invece di 1 e 0 e fornisce chiari messaggi di errore.

2. Input robusto: non risponde a qualsiasi carattere salvo quelli legittimi.

Sezione B: Output su schermo

3. Grandi lettere: visualizza il testo in formato quattro volte il normale.
4. Output formattato: visualizza numeri con numero di cifre decimali specificato dall'utente.
5. Visualizzazione di stringhe: visualizza una lunga stringa senza interrompere le parole sulle righe.
6. Convertitore binario: visualizza un numero in binario.

Sezione C: Manipolazione interna

7. Estrazione del cognome: estrae il cognome da una stringa che contiene il nome completo di una persona.

8. Ricerca di liste: analizza una lista alla ricerca di un'entrata specifica.
9. Bubble sort: riordina un piccolo numero di stringhe in ordine alfabetico.
10. Quick sort: riordina una lista di numeri (o stringhe).

Sezione D: Matematica

11. Semplificatore di frazioni: riduce la frazione ai suoi minimi termini.
12. Equazioni simultanee: risolve le equazioni simultanee.

1. INPUT TOLLERANTE

Scopo: Consentire ad un utente inesperto di immettere numeri. Tutti gli spazi sono ignorati e le lettere I e O sono intese rispettivamente come cifre 1 e 0. Tutti gli altri errori sono chiaramente spiegati.

righe: 4500-4660

Parametri: Output: il risultato è fornito in X1.

Variabili locali: XX$, YY$, JJ, CC$

```
4500 REM INPUT TOLLERANTE DI NUMERI
4510 INPUT XX$
4520 YY$=" "
4530 FOR JJ - 1 TO LEN (XX$)
4540 CC$ = MID$ (XX$,JJ,1)
4550 IF CC$ = "O" THEN YY$ = YY$ + "0":
     GOTO 4600 : REM SOSTITUISCI
     LETTERA O CON CIFRA 0
4560 IF CC$ = "I" THEN YY$ = YY$ + "1"
     GOTO 4600
4570 IF CC$ = " " THEN 4600
4580 IF NOT (CC$<= "9" AND CC$ > =
     "0" OR CC$ = "+" OR CC$ = "-"
     OR CC$= ".") THEN 4620
4590 YY$ = YY$ + CC$
4600 NEXT JJ
4610 X1 = VAL(YY$) : RETURN
4620 PRINT "NUMERI CONSISTONO IN"
```

```
4630 PRINT "CIFRE DECIMALI 0-9,"
4640 PRINT "+, - E . SOLTANTO"
4650 PRINT "PROVA ANCORA"
4660 GOTO 4510
```

Programma di gestione:

```
10 GOSUB 4500
20 PRINT "VALORE ="; X1
30 GOTO 10
```

Risultati della prova:

```
RUN
? 778
VALORE = 778
? IOI
VALORE = 101
? -34.56
VALORE = -34.56
? 45.K
NUMERI CONSISTONO IN
CIFRE DECIMALI 0-9,
+, - E . SOLTANTO
PROVA ANCORA
? 7.7
VALORE = 7.7
```

2. INPUT ROBUSTO

Scopo: Leggere un numero dalla tastiera, ignorando tutti i caratteri privi di significato. Può essere usato DEL e un numero è sempre terminato da RETURN;

Righe: 7000-7090

Parametri: Output: Risultato fornito in X1.

Variabili locali: PP, AA$, XX$

```
7000 REM INPUT NUMERI ROBUSTO
7010 XX$ = " ":P = 0
7020 GEST AA$ : IF AA$ = " " THEN 7020
7030 IF AA$ > = "0" AND AA$ < = "9"
     THEN PRINT AA$;: XX$ = XX$ + AA$:
     PP = PP + 1 : GOTO 7020
7040 IF ASC(AA$) <> 20 THEN 7070:
     REM CERCA DEL
7050 IF PP=0 THEN 7020: REM NON
     POSSO CANCELLARE NULLA
7060 PRINT "[SHIFT] e [CRSR] spazio
     [SHIFT] e [CRSR]";: PP=PP-1:
     XX$=LEFT$(XX$,PP) : GOTO 7020
7070 IF ASC (AA$)<> 13 THEN 7020 : REM
     CERCA RETURN
7080 IF PP=0 THEN 7020 : REM DEVE ESSERE
     UNA QUALCHE CIFRA
7090 X1=VAL (XX$) : RETURN
```

Programma di gestione:

```
10 GOSUB 7000
20 PRINT X1
30 GOTO 10
```

3. BIG LETTERS (GRANDI LETTERE)

Scopo: Visualizzare i caratteri **64** quattro volte la loro dimensione abituale.

Righe: da 8000 a 8200

Parametri Input: Il successivo carattere da visualizzare è fornito in A1$. Può essere qualsiasi carattere stampabile o spazio, RETURN, CLR/HOME, un codice di colore, CTRL e RVS ON o CTRL e RVS OFF.

Variabili locali: AA,BB,JJ,KK,LL,MM,NN,QQ.

Nota: QQ tiene nota dello stato RVS corrente e non deve essere usato al difuori della subroutine.

```
8000 REM VISUALIZZA I CARATTERI IN A1$
     4 VOLTE PIÙ GRANDI
8010 BB = ASC (A1$)
8020 IF BB = 13 OR BB = 141 THEN PRINT
     "[CRSR][CRSR][CRSR]": RETURN
8030 IF BB = 18 THEN QQ = 1: RETURN
8040 IF BB = 146 THEN QQ = 0: RETURN
8050 IFBB<32THENPRINTMID$("
```

```
     ",BB+1,1);:RETURN
8060 IFBB>=144 AND BB<160THENPRINT
     MID$("
```

```
     ",BB-143,1);: RETURN
8070 AA= (BB AND 31)+0.5 * (BB AND
     128): IF (BB AND 64)=0 THEN
     AA=AA+32
8080 FOR JJ=0 TO 6 STEP 2
8085 POKE 56334,PEEK(56334) AND 254: POKE
     1,PEEK(1) AND 251
8090 KK=PEEK (53248+8*AA+JJ):
     LL=PEEK(53249+8*AA+JJ)
8095 POKE 1,PEEK(1) OR 4: POKE 56334, PEEK
     (56334) OR 1
8100 NN=64: FOR MM=0 TO 3
```

```
8110 PP = 1+8★ INT (KK/NN)+2★INT
      (LL/NN)
8120 KK=KK−INT (KK/NN)★NN:LL=LL−
      INT (LL/NN)★NN

8130 IFQQ=ØTHENPRINTMID$(" [CTRL]
e [RVS OFF] [SPACE] [CTRL] e [RVS OFF]
[C=] e [D] [CTRL] e [RVS OFF] [C=]
e [F] [CTRL] e [RVS OFF] [C=] e
[I] [CTRL] e [RVS OFF] [C=] e [C]
[CTRL] e [RVS ON] [C=] e [K]
[CTRL] e [RVS ON] [C=] e [B]
[CTRL] e [RVS ON] [C=] e [V]
[CTRL] e [RVS OFF] [C=] e [V]
[CTRL] e [RVS OFF] [C=] e [B]
[CTRL] e [RVS OFF] [C=] e [K]
[CTRL] e [RVS ON] [C=] e [C]
[CTRL] e [RVS ON] [C=] e [I]
[CTRL] e [RVS ON] [C=] e [F]
[CTRL] e [RVS ON] [C=] e [D]
[CTRL] e [RVS ON] [SPACE]

",PP,2);:GOTO815Ø

814Ø PRINTMID$(" [CTRL] e [RVS ON]
[SPACE] [CTRL] e [RVS ON] [C=] e [D]
[CTRL] e [RVS ON] [C=] e [F]
[CTRL] e [RVS ON] [C=] e [I]
[CTRL] e [RVS ON] [C=] e [C]
[CTRL] e [RVS OFF] [C=] e [K]
[CTRL] e [RVS OFF] [C=] e [B]
[CTRL] e [RVS OFF] [C=] e [V]
[CTRL] e [RVS ON] [C=] e [V]
[CTRL] e [RVS ON] [C=] e [B]
[CTRL] e [RVS ON] [C=] e [K]
[CTRL] e [RVS OFF] [C=] e [C]
[CTRL] e [RVS OFF] [C=] e [I]
[CTRL] e [RVS OFF] [C=] e [F]
[CTRL] e [RVS OFF] [C=] e [D]
[CTRL] e [RVS OFF] [SPACE] ",PP,2);
815Ø NN=INT(NN/4):NEXT MM
816Ø PRINT" [CRSR ↑↓] [SHIFT] e [CRSR ⇐⇒] 4 volte";
817Ø NEXTJJ
818Ø PRINT" [SHIFT] e [CRSR ↑↓] 4 volte [CRSR ⇐⇒]
      4 volte";
819Ø IFPEEK(211)>36THENPRINT" [CRSR ↑↓]
      2 volte"
82ØØ RETURN
```


Programma di gestione

```
1Ø GET A1$ : IF A1$= " " THEN 1Ø
2Ø GOSUB 8ØØØ
3Ø GOTO 1Ø
```

Passata campione: Non riproducibile. (Provare per credere).

4. NUMERO FORMATTATO

Scopo: Visualizzare un numero in un formato controllato.

Righe: da 5ØØØ a 513Ø

Parametri: Input: Numero da visualizzare in X1
Numero di cifre decimali richieste in Y1.

Variabili locali: NN$, PP, JJ, XX

Nota: Se il numero da visualizzare è maggiore di 999999999, viene visualizzato in virgola mobile senza arrotondamento, altrimenti è arrotondato e visualizzato con Y1 cifre decimali dopo il punto decimale. Se Y1 = Ø, il numero è arrotondato all'intero più vicino.

```
5ØØØ REM VISUALIZZA DA X1 a Y1 CIFRE
      DECIMALI
5Ø1Ø XX=X1: IF Y1 >Ø AND ABS(XX) <=
      999999999 THEN 5Ø5Ø
5Ø2Ø XX=XX+0·5
5Ø3Ø PRINT INT (XX);
5Ø4Ø RETURN
5Ø5Ø IF XX <Ø THEN XX=XX−Ø·5★1Ø↑−
      Y1:GOTO 5Ø7Ø
5Ø6Ø XX=XX+Ø·5★1Ø↑−Y1
5Ø7Ø NN$= STR$ (XX)
5Ø8Ø FOR PP = 1 TO LEN (NN$)
5Ø9Ø IF MID$(NN$, PP, 1) = "." THEN
      PRINT LEFT$(NN$, PP+Y1);: RETURN
51ØØ NEXT PP
511Ø PRINT NN$; ".";
512Ø FOR JJ=1 TO Y1: PRINT "Ø";: NEXT JJ
513Ø RETURN
```

Programma di gestione

```
10 FOR J= 667 TO 670
20 FOR Y1 =0 TO 3
30 X1 = SQR(J)
40 GOSUB 5000
50 NEXT Y1
60 PRINT
70 NEXT J
80 STOP
```

Passata campione:

26	25·8	25·83	25·826
26	25·8	25·85	25·846
26	25·9	25·87	25·865
26	25·9	25·88	25·884

5. VISUALIZZAZIONE DI STRINGHE

Scopo: Visualizzare una stringa senza dividere le parole sulle righe.

Righe: 5700-5800

Parametri: Input: stringa da visualizzare in X1$.

Variabili locali: XX$, PP, QQ, RR.

```
5700 REM VISUALIZZA X1$ SENZA
     DIVIDERE LE PAROLE
5710 XX$=X1$
5720 PP=LEN(XX$)
5730 IF PP <=40 THEN RR=PP:GOSUB
     5780: RETURN
5740 FOR QQ=41 TO 1 STEP -1
5750 IF MID$(XX$,QQ,1) = " " THEN
     RR=QQ-1:GOSUB 5780: XX$=
     RIGHT$(XX$, PP-QQ):GOTO
     5720
5760 NEXT QQ
5770 RR=40:GOSUB 5780: XX$=RIGHT$
     (XX$, PP-40): GOTO 5720
5780 REM SUBROUTINE INTERNA
5790 PRINT LEFT$(XX$,RR);: IF RR<40
     THEN PRINT
5800 RETURN
```

Programma di gestione

```
10 X1$="THIS IS THE FIRST PART
   OF A LONG STRING TO TRY OUT "
20 X1$=X1$+"THE STRING PRINTING
   ROUTINE. IT SHOULD ARRANGE "
30 X1$=X1$+"THE WORDS PROPERLY"
40 GOSUB 5700
50 STOP
```

Output campione: provare e vedere!

6. CONVERTITORE BINARIO

Scopo: Visualizzare il profilo binario di un numero nel campo da 0 a 255.

Righe: da 1000 a 1060

Parametri: Input: Numero da visualizzare in X1

Variabili locali: YY, KK, XX.

```
1000 REM CONVERTI X1 IN BINARIO E
     VISUALIZZA
1010 YY=256: XX=X1: FOR KK = 1 TO 8
1020 YY=YY/2
1030 IF XX>= YY THEN XX=XX-YY:
     PRINT"★";:GOTO 1050
1040 PRINT" ";
1050 NEXT KK
1060 PRINT:RETURN
```

Programma di gestione:

```
20 FOR J=0 TO 9
20 X1=J: GOSUB 1000
30 NEXT J
40 STOP
```

Output campione:

7. ESTRAZIONE DEL COGNOME

Scopo: Estrarre un cognome dal nome intero di una persona.

Righe: da 4100 a 4200

Parametri: Input: Il nome di una persona, in NI$, in una qualsiasi delle seguenti forme:
J.X. SMITH
HEORGE ELLIOT
ALVA T EDISON
WELLINGTON-COO
K.O'SHAUGNESSY

Output: Il cognome della persona in YI$. Il cognome è definito con una sequenza ininterrotta di lettere, trattini e apostrofi vicino alle estremità della stringa in NI$.

Esempi sono: SMITH
ELLIOT
EDISON
WELLINGTON-COO
O'-SHAUGHESSY

Se il cognome non può essere trovato, YI$ è fornito vuoto.

Variabili locali: JJ, KK, CC$

Note: Questa subroutine funziona correttamente per i nomi europei ma richiede modifiche per i nomi di altre parti del mondo — ad esempio Cina.

```
41100 REM ESTRAI COGNOME DA NI$ E
      PRESENTALO IN YI$
4110 JJ=LEN (NI$)
4120 IF JJ=0 THEN YI$= "": RETURN
4130 IF MID$(NI$, JJ, 1) <"A" OR MID$
     (NI$, JJ, 1) > "Z" THEN JJ=JJ-1:
     GOTO 4120
4140 FOR KK=JJ TO 1 STEP -1
4150 CC$=MID$(NI$, KK, 1)
4160 IF NOT (CC$ >= "A" AND CC$ <=
     "Z" OR CC$= "-" OR CC$= "'")
     THEN 4190
4170 NEXT KK
4180 KK=0
4190 YI$=MID$(NI$, KK+1, JJ -KK)
4200 RETURN
```

Programma di gestione:

```
10 INPUT "NOME PREGO"; NI$
20 GOSUB 4100
30 PRINT "IL COGNOME È"; YI$
40 GOTO 10
```

Passata campione:

```
NOME PREGO? J. X. SMITH
IL COGNOME È SMITH
NOME PREGO? GEORGE ELLIOT
IL COGNOME È ELLIOT
NOME PREGO? WELLINGTON-COO
IL COGNOME È WELLINGTON-COO
NOME PREGO? K. O'SHAUGHNESSY
IL COGNOME È O'SHAUGHNESSY
```

8. RICERCA IN LISTE

Scopo: Cercare una lista ordinata, un'entrata specifica, usando il metodo della ricerca dicotomica

Righe: 6000-6050

Parametri: Input: La Lista da esaminare (deve essere in ordine alfabetico o crescente) in AI$.
Indice della prima entrata in L1; indice dell'ultima entrata in H1.
Stringa da osservare in XI$.

Output: Se l'entrata viene trovata, M1 contiene il suo indice. Se non è trovato M1=−1.

Variabili locali: HH, LL.

```
6000 REM CERCA IN LISTA ORDINATA AI$
6005 HH=H1 : LL=L1
6010 IF HH<LL THEN M1 =-1 : RETURN
6020 M1 = INT (0·5*(HH+LL))
6030 IF XI$= AI$(M1) THEN RETURN
6040 IF XI$ < AI$(M1) THEN HH=M1 -1 :
     GOTO 6010
6050 LL=M1+1 : GOTO 6010
```

Programma di gestione:

```
10 DATA ABLE, BAKER, CHARLIE, DOG,
   ERNIE, FRED, GORDON
20 DATA HARRY, IONA, JILL, KATE, LYDIA,
   MURIEL, NICK
30 DIM AI$(14)
40 FOR J=1 TO 14 : READ AI$(J) : NEXT J
50 INPUT"BATTI UN NOME";XI$
60 H1=14:L1=1:GOSUB 6000
70 IF M1<0 THEN PRINT "NON TROVATO":
   GOTO 50
80 PRINT"TROVATO ALL'ENTRATA";
   M1 : GOTO 50
```

Passata campione:

```
BATTI UN NOME? CHARLIE
TROVATO ALL'ENTRATA 3
BATTI UN NOME? DAVID
NON TROVATO
BATTI UN NOME? NICK
TROVATO ALL'ENTRATA 14
...
```

9. BUBBLE SORT

Scopo: Riordinare poche voci di una stringa in ordine alfabetico.

Righe: 6500-6560

Parametri: Input: Elenco di voci da riordinare in AI$(1) fino a AI$(N1).
Numero degli elementi in N1.

Output: Lista ordinata da AI$(1) fino a AI$(N1).

Variabili locali: KK, DD$, SS$

```
6500 REM BUBBLE SORT DI STRINGHE IN AI$
6510 SS$="NO"
6520 FOR KK=1 TO N1-1
```

```
6530 IF AI$(KK)>AI$(KK+1) THEN DD$=
     AI$(KK): AI$(KK)=AI$(KK+1):
     AI$(KK+1)=DD$: SS$= "YES"
6540 NEXT KK
6550 IF SS$= "YES" THEN 6510
6560 RETURN
```

Programma di gestione:

```
10 INPUT "N1"; N1
20 PRINT"BATTI"; N1; "PAROLE"
30 DIM AI$(N1)
40 FOR J=1 TO N1 : INPUT AI$(J) : NEXT J
50 GOSUB 6500
60 FOR J=1 TO N1 : PRINT AI$(J) : NEXT J
70 STOP
```


Passata campione:

```
RUN
N1 ?7
BATTI 7 PAROLE
? PEARS
? CHERRIES
? BANANAS
? ORANGES
? DATES
? PLUMS
? APPLES
APPLES
BANANAS
CHERRIES
DATES
ORANGES
PEARS
PLUMS
```

10. QUICK SORT

Scopo: Riordinare elementi in ordine numerico usando l'algoritmo Quick Sort di Hoare.

Numeri di riga: da 6200 a 6380

Parametri: Input: Lista dei numeri da riordinare da A1(1) fino a A1(N1).

Output: La lista riordinata compare da A1(1) fino a A1(N1).

Variabili locali: SS, SS%, AA, BB, XX, YY, ZZ, DD, PP.

Note: *(i)* SS non deve essere usato altrove nel programma se la subroutine di riordino è usata più di una volta.

(ii) La subroutine riordinerà stringhe invece di numeri se vengono effettuate le seguenti sostituzioni:
AI$ per A1; ZZ$ per ZZ;
DD$ per DD

(iii) Se la routine viene usata per riordinare una serie di record con i campi che abbracciano parecchie matrici, le seguenti istruzioni devono essere modificate per assicurare che tutti i campi di ogni record siano spostati:
6280 6290

L'operazione di confronto nelle istruzioni 6260 e 6270, può essere invertita o sostituita se si richiede un diverso ordinamento.

```
6200 REM QUICKSORT
6210 IF SS=1 THEN 6230
6220 DIM SS%(100):SS=1: REM DECLARE
     STACK
6230 AA=1: BB=N1: SS%(0)=1: PP=1
6240 XX=AA: YY=BB: ZZ=A1(BB)
6250 IF XX >= YY THEN 6290
6260 IF A1(XX) <= ZZ THEN XX=XX+1:
     GOTO 6250
6270 IF A1(YY) >= ZZ THEN YY=YY-1:
     GOTO 6250
6280 DD=A1(YY): A1(YY)=A1(XX): A1(XX)=
     DD: GOTO 6250
6290 A1(BB)=A1(XX): A1(XX)=ZZ
6300 IF XX-AA <=1 THEN 6340
6310 SS%(PP)=XX: SS%(PP+1)=BB:
     SS%(PP+2)=2: PP=PP+3
6320 BB=XX-1: GOTO 6240
6330 PP=PP-3: XX=SS%(PP): BB=SS%
     (PP+1)
6340 IF BB-XX <=1 THEN 6370
6350 SS%(PP)=3: PP=PP+1: GOTO 6240
6360 PP=PP-1
6370 ON SS%(PP-1) GOTO 6380, 6330,
     6360
6380 RETURN
```

Programma di gestione:

```
10 INPUT "QUANTI";N1
20 DIM A1 (N1)
30 FOR J=1 TO N1:A1(J)=INT(1000*RND(0))
   : NEXT J
40 GOSUB 6200
50 FOR J=1 TO N1:PRINT A1(J);: NEXT J
60 STOP
```

Passata campione: QUANTI? 6
331 342 369 540 870 912
(cioè 6 numeri in ordine ascendente)

11. SEMPLIFICATORE DI FRAZIONI

Scopo: Ridurre le frazioni ai rispettivi minimi termini.

Numeri di riga: da 5500 a 5630

Parametri: Input: *A1 (Numeratore della frazione)*
B1 (Denominatore di frazione)

Output: C1 (Numeratore di frazione semplificata)
D1 (Denominatore di frazione semplificata)

Variabili locali: JJ, KK

```
5500 REM RIDUCI FRAZIONE A1/B1 AI SUOI
     MINIMI TERMINI
```

```
5510 REM RISULTATO IN C1/D1. LOCALI
     SONO JJ, KK
5520 REM ERRORE SE A1 O B1 NON SONO
     NUMERI INTERI O SE B1 < 1
5530 IF A1=INT(A1)AND B1=INT(B1)
     AND B1 > 1 THEN 5550
5540 PRINT"PARAMETRI ERRATI PER
     SEMPLIFICATORE FRAZIONE";A1;B1:STOP
5550 IF A1=0 THEN C1=0:D1=1:RETURN
5560 JJ=A1:KK=B1
5570 IF A1 < 0 THEN JJ=- A1
5580 IF KK=0 THEN 5620
5590 IF JJ=0 THEN JJ=KK:GOTO 5620
5600 JJ > KK THEN JJ=JJ-INT(JJ/KK) ★
     KK:GOTO 5580
5610 KK=KK-INT  (KK/JJ)★JJ:GOTO 5580
5620 C1=A1/JJ:D1=B1/JJ
5630 RETURN
```

Programma di gestione:

```
10 INPUT "INDICA FRAZIONE"; A1,B1
20 GOSUB 5500
30 PRINT C1;"/"; D1
40 GOTO 10
```

Passata campione:

```
RUN
INDICA FRAZIONE? 2, 4
1/2
INDICA FRAZIONE? 123, 456
41/152
INDICA FRAZIONE? 375, 1000
3/8
INDICA FRAZIONE? 0, 1234
0/1
INDICA FRAZIONE? 0.5, 0.75
PARAMETRI ERRATI PER SEMPLIFICATORE
FRAZIONE .5
.75
RUN
INDICA FRAZIONE? -50, 100
-1/2
INDICA FRAZIONE? 77,0
PARAMETRI ERRATI PER SEMPLIFICATORE
FRAZIONE 77
0
```

12. EQUAZIONI SIMULTANEE

Scopo: Risolvere equazioni simultanee, ad esempio N1 equazioni in N1 incognite

Righe: da 9000 a 9270

Parametri: Input: N1: il numero di equazioni da A1(1,1) a A1(N1,N1); una matrice bidimensionale che contiene la matrice dei coefficienti da B1(1) a B1(N1): il vettore dei membri di destra.
Output: da X1(1) a X1(N1) contiene il vettore delle soluzioni.

Variabili locali: DD, JJ, KK, LL

Nota: I valori iniziali delle matrici A1 e B1 sono distrutti.

Esempio: Si considerino tre equazioni in tre incognite:

$$3x + 2y + 1z = 19$$
$$2x + 7y + 2z = 55$$
$$4x + 1y + 4z = 19$$

Un programma, per risolvere questa equazione, potrebbe essere scritto come segue:

```
10 DATA 3, 2, 1, 19
20 DATA 2, 7, 2, 55
30 DATA 4, 1, 4, 19
40 DIM A1(3,3), B1(3), X1(3)
50 N1=3
60 FOR j=1 TO N1: FOR K=1 TO N1:
   READ A1 (K,J): NEXT K: READ B1(J):
   NEXT J
70 GOSUB 9000: REM RICHIAMA
   SUBROUTINE
80 FOR J=1 TO N1: PRINT X1(J):
   NEXT J
90 STOP
```

Tempo: Il tempo richiesto per risolvere una serie di N1 equazioni è all'incirca proporzionale al cubo di N1. Valori tipici sono:

N1	Tempo (secondi)
5	2
10	10
15	30
20	65
25	121

```
9000 REM RISOLVI N1 EQUAZIONI
     SIMULTANEE A1.X1=B1
9010 IF N1=1 THEN X1(1)=B1(1)/A1(1,1):
     RETURN
9020 FOR JJ=1 TO N1-1:REM TROVA PIVOT
9030 DD=ABS(A1(JJ,JJ)):LL=JJ
9040 FOR KK=JJ TO N1
9050 IF ABS(A1(KK,JJ)) > DD THEN DD=
     ABS(A1(KK,JJ)): LL=KK
9060 NEXT KK
9070 IF LL=JJ THEN 9120
9080 FOR KK=JJ TO N1: REM INTERSCAMBIA
     EQUAZIONI
9090 DD=A1(JJ,KK): A1(JJ,KK)=A1(LL,KK):
     A1(LL,KK)=DD
9100 NEXT KK
9110 DD=B1 (JJ): B1 (JJ)=B1 (LL): B1 (LL)=DD
9120 FOR KK=JJ+1 TO N1: DD=A1(KK,JJ)/A1
     (JJ,JJ)
9130 FOR LL=JJ TO N1 : REM ELIMINA
9140 A1 (KK,LL)=A1 (KK,LL)-DD★A1 (JJ,LL)
9150 NEXT LL
9160 B1 (KK)=B1 (KK)-DD★B1 (JJ)
9170 NEXT KK
9180 NEXT JJ
9190 FOR JJ=N1 TO 1 STEP-1 : REM
     RISOSTITUISCI
9200 DD=B1(JJ)
9210 IF JJ=N1 THEN 9250
9220 FOR KK=JJ+1 TO N1
9230 DD=DD-X1(KK)★A1 (JJ,KK)
```

```
9240 NEXT KK
9250 X1 (JJ)=DD/A1(JJ,JJ)
9260 NEXT JJ
9270 RETURN
```

Programma di gestione:

```
 10 INPUT "N1";N1
 20 DIM A1(N1,N1), B1(N1), X1(N1), Y1(N1)
 30 FOR J=1 TO N1
 40 FOR K=1 TO N1
 50 A1(J,K)= 100 ★(RND(0)−0·5)
 60 NEXT K
 70 PRINT "Y1(";J; ")";: INPUT Y1(J)
 80 NEXT J
 90 FOR J=1 TO N1
100 B1(J)=0
110 FOR K=1 TO N1
120 B1(J)= B1(J)+Y1(K)★A1(J,K)
130 NEXT K,J
140 X=TI
150 GOSUB 9000
160 X=TI−X
170 FOR J=1 TO N1
180 PRINT Y1(J);X1(J)
190 NEXT J
200 PRINT "TEMPO="; INT (X/600.5)
210 STOP
```


NOTA SUL PROGRAMMA DI GESTIONE

Questo programma è studiato per trovare la subroutine per la risoluzione di equazioni simultanee e per presentare i suoi risultati in una forma che possa essere facilmente controllata.
Il programma inizia chiedendo all'utente il numero di equazioni da risolvere. Richiede quindi un numero appropriato di valori per le "incognite".
Successivamente, il programma costruisce una serie di equazioni per le incognite usando coefficienti casuali. Esso le risolve e presenta una serie di risultati unitamente ai valori originali. I risultati dovrebbero essere gli stessi, salvo piccoli errori di arrotondamento.

Passata campione:

```
RUN
N1? 1
Y1(1)? 6
  6 6                (Uguale!)
TEMPO = 0
```

```
RUN
N1? 4
Y1(1)? 4
Y1(2)? 1
Y1(3)? 7
Y1(4)? 8
  4 4
  1 1                       Uguale
  7 7
  8 8.000000001      Circa uguale
TEMPO = 0
```

UNITA': 16

Esperimento 16.1

```
  10 INPUT"STIPENDI IN CENTS";W
  20 FOR J=1 TO 8
  30 READ V,N$
  40 T=INT(W/V)
  50 PRINT T;N$
  60 W=W-V★T
  70 NEXTJ
  80 STOP
1000 DATA 5000, BANCONOTE 50 DOLLARI
1010 DATA 1000, BANCONOTE 10 DOLLARI
1020 DATA 500, BANCONOTE DA 5 DOLLARI
1030 DATA 100, DOLLARI
1040 DATA 25, QUARTI DI DOLLARO
1050 DATA 10, DECIMI DI DOLLARO
1060 DATA 5,5 CENTS
1070 DATA 1,CENTS
```

Esperimento 16.2A

```
  10 FOR K=1 TO 12
  20 READ M$
  30 PRINT M$
  40 NEXT K
  50 STOP
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1020 DATA SETTEMBRE, OTTOBRE, NOVEMBRE,
     DICEMBRE
```

Esperimento 16.2B

```
  10INPUT G,M,A
  20 FOR J=1 TO M
  30 READ M$
  40 NEXT J
  50 PRINT G;M$;A
  60 RESTORE
  70 GOTO 10
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1020 DATA SETTEMBRE, OTTOBRE, NOVEMBRE,
     DICEMBRE
```


Esperimento 16.3

```
 10 T=0
 20 S=0
 30 PRINT"[SHIFT] e [CLR HOME]"
 40 READA$
 50 IF A$="END" THEN 240
 60 READ B$
 70 T=T+1
 80 J=1
 90 PRINTA$
100 PRINT
110 INPUT X$
120 IF X$=B$ THEN 200
130 IF J=3 THEN 170
140 J=J+1
150 PRINT"ERRATO. RIPROVA"
160 GOTO 90
170 PRINT"RISPOSTA="
180 PRINT B$
190 GOTO 40
200 PRINT"ESATTO!"
210 IF J > 1 THEN 40
220 S=S+1
230 GOTO 40
240 PRINT"HAI RISPOSTO";S;"ESATTO"
250 PRINT"AL PRIMO COLPO"
260 PRINT"SU";T;"DOMANDE"
270 STOP
280 DATA CHI COMPOSE IL MESSIA, HANDEL
290 DATA QUANTE SONO LE SINFONIE
    SCRITTE DA BEETHOVEN, NOVE
300 DATA CHI HA SCRITTO LA CARMEN, BIZET
310 DATA COSA SUONAVA PAGANINI,
    VIOLINO
320 DATA END
```

UNITA': 17

Esperimento 17.1A

```
10 INPUT"QUANTI MINUTI";M:R=TI+M
   ★3600
20 IF TI < R THEN 20
30 PRINT"TEMPO SCADUTO":STOP
```

Esperimento 17.1B

```
10 PRINT"USA 1000000 PER":PRINT"
   TERMINARE INPUT":S=0:N=0
20 INPUT"NUMERO SUCCESSIVO";X:IF X =
   1000000 THEN 40
30 S=S+X:N=N+1:GOTO 20
40 PRINT"MEDIA=";S/N:STOP
```

Esperimento 17.1C

```
 10 REM DEBUG THIS PROGRAM!
 20 INPUT"NAME";N$
 30 IFN$="JIM"THEN A$="JAMES":GOTO
    100
 40 IFN$="BOB"THEN A$="ROBERT":GOTO
    100
 50 IFN$="KATE"THEN A$="KATHERINE":
    GOTO100
 60 IFN$="PENNY"THEN A$="PENELOPE":
    GOTO100
 70 PRINT N$;" IS NOT"
 80 PRINT"SHORT FOR ANYTHING."
 90 GOTO10
100 PRINT N$;" IS SHORT"
110 PRINT"FOR ";A$
120 GOTO10
```

Esperimento 17.2A

```
N$<>"JONES" AND N$<>"SMITH"
AND N$<>"BROWN"
X>15 OR X<4
```

Esperimento 17.2B

```
10 REM ESERCIZIO 17.2C
20 IF T < 0.1 THEN
   PRINT"FANTASTICO!!": GOTO 100
30 IF T < 0.15 THEN
   PRINT"ECCEZIONALE!": GOTO 100
40 IF T > = 0.15 AND T < 0.2 THEN
   PRINT"MOLTO BUONO":GOTO 100
50 IF T > = 0.2 AND T < 0.25 THEN
   PRINT"BUONO":GOTO 100
60 IF T > = 0.25 AND T < 0.28 THEN
   PRINT"DISCRETO":GOTO 100
70 IF T > = 0.28 AND T < 0.33 THEN
   PRINT"MOLTO LENTO": GOTO 100
80 IF T > = 0.33 AND T < 0.4 THEN
   PRINT"SVEGLIA!":GOTO 100
90 IF T > 0.4 THEN PRINT"PROVA
   ANCORA QUANDO SEI SOBRIO!!"
100 STOP
```


Esperimento 17.2C

```
10 INPUT"PAROLA";W$
20 IF W$ > ="ABRAHAM" AND W$ < =
   "FRANCE"' THEN PRINT"USA VOL 1":STOP
30 IF W$ > ="FRANCHISE" AND W$ < =
   "LEVANTE" THEN PRINT"USA VOL 2":STOP
40 IF W$ > ="LEVITAZIONE" AND W$ < =
   "QUOTA" THEN PRINT"USA VOL 3":STOP
50 IF W$ > ="QUOZIENTE" AND W$ < =
   "XILOFONO" THEN PRINT"USA VOL 4":
   STOP
60 PRINT"QUESTA PAROLA NON È"
70 PRINT"NELL'ENCICLOPEDIA"
80 STOP
```

UNITA': 18

Esperimento 18.1A

```
  10 PRINT " [SHIFT] and [CLR HOME] ARITHMETIC
     TEST"
  20 PRINT "ANSWER THE FOLLOWING
     SUMS"
  30 S=0
  40 FOR A=1 TO 10
  50 X = INT (10★RND(0)+1)
  60 Y = INT (10★RND(0)+1)
  70 PRINT X; "+"; Y; "=";
  80 INPUT Z
  90 GOSUB 1000:REM MAKE SOUND
 100 GOSUB 2000:REM CHANGE BORDER
     COLOUR
 110 NEXT A
 120 PRINT "THAT'S"; S; "RIGHT OUT OF 10"
 130 FOR R=1 TO S
 140 GOSUB 1000:REM MAKE SOUND
 150 NEXT R
 160 STOP
1000 REM SUBROUTINE TO MAKE PIP SOUND
1010 VV = 212★256
1020 POKE VV + 24, 15: POKE VV,0
1030 POKE VV+1,80: POKE VV+5,9
```

```
1040 POKE VV+4,0: POKE VV+4,33
1050 FOR MM = 1 TO 800: NEXT MM
1060 RETURN
2000 REM SUBROUTINE TO CHANGE BORDER
     COLOUR
2010 IF Z=X+Y THEN POKE 53280,4: S=S+1:
     GOTO 2030:REM RIGHT ANSWER —
     BORDER PURPLE
2020 POKE 53280,0: REM WRONG ANSWER
     — BORDER BLACK
2030 FOR KK=1 TO 800:NEXT KK
2040 POKE 53280,14: REM RESTORE INITIAL
     COLOUR
2050 RETURN
```

Esperimento 18.1B

```
  10 PRINT" SHIFT and CLR HOME COUNTING
     TEST"
  20 PRINT"COUNT THE PIPS"
  30 S=0
  40 FOR A=1 TO 10
  50 FOR T=1 TO 5000:NEXT T:REM WAIT A BIT
  60 X=INT(9*RND(0)+1)
  70 FOR J=1 TO X
  80 GOSUB 1000
  90 NEXT J
 100 INPUT Z
 110 IF Z=X THEN GOSUB 5000:GOSUB 2000:
     GOSUB 4000:GOTO130
 120 GOSUB3000:GOSUB 4000
 130 NEXT A
 140 PRINT"THAT'S";S;"RIGHT"
 150 STOP
1000 REM SUBROUTINE TO MAKE PIP SOUND
1010 VV = 212*256
1020 POKE VV + 24, 15: POKE VV,0
1030 POKE VV+1,80: POKE VV+5,9
1040 POKE VV+4,0: POKE VV+4,33
1050 FOR MM = 1 TO 800: NEXT MM
1060 RETURN
2000 REM SUBROUTINE TO CHANGE BORDER
     COLOUR PURPLE—RIGHT ANSWER
2010 IFZ=XTHENPOKE53280,4:S=S+1:
     RETURN
3000 REM SUBROUTINE TO CHANGE BORDER
     COLOUR BLACK—WRONG ANSWER
3010 POKE 53280,0:RETURN
4000 REM SUBROUTINE TO RESTORE INITIAL
     COLOUR
4010 FORKK=1TO800:NEXTKK
4020 POKE 53280,14:RETURN
5000 REM MYSTERY
5010 VV=212*256: POKE VV+24,15
5020 POKE VV,0: POKE VV+5,45
5030 POKE VV+4,0: POKE VV+4,33
5040 FOR MM=50 TO 120 STEP 0.5
5050 POKE VV+1,MM
5060 NEXT MM
5070 POKE VV+24,0: RETURN
```

Esperimento 18.2

Esperimento 18.3

```
10 PRINT" [SHIFT] e [CLR HOME] "
20 X1=1:Y1=22:N1=17:GOSUB2000
30 X1=1 :Y1=5:N1=3:GOSUB3000
40 X1=4:Y1=3:N1=3:GOSUB4000
50 X1=7:Y1=6:N1=5:GOSUB2000
60 X1=7:Y1=10:N1=13:GOSUB1000
70 X1=20:Y1=10:N1=11:GOSUB2000
80 X1=1:Y1=21:N1=20:GOSUB1000
90 X1=1:Y1=21:N1=20:GOSUB1000
100 X1=3:Y1=5:GOSUB5000
110 Y1=15
120 FORX1=2 TO 19 STEP 4
130 GOSUB6000
140 NEXT X1
150 X1=4:Y1=0:GOSUB7000:REM DRAW
      CROSS
160 GOTO160
500 REM POSITION CURSOR TO X1,Y1
510 PRINT" [CLR HOME] ";
520 IF X1=0 THEN 540
530 FORKK=1 TO X1:PRINT" [CRSR ⇐⇒] ";:NEXTKK
540 IF Y1=0 THEN RETURN
550 FORKK=1 TO Y1:PRINT" [CRSR ⇑⇓] ";:NEXTKK
560 RETURN
1000 REM TO DRAW HORIZONTAL LINE FOR
      N1 UNITS FROM X1,Y1
1010 GOSUB 500:REM POSITION CURSOR
1020 FOR JJ=1 TO N1
1030 PRINT" [CTRL] e [RVS ON] [SPACE] ";
1035 NEXT JJ
1037 PRINT" [CTRL] e [RVS OFF] ";
1040 RETURN
2000 REM DRAW VERTICAL LINE N1 UNITS
      DOWN FROM X1,Y1
2010 GOSUB 500:REM POSITION CURSOR
2020 FOR JJ=1 TO N1
2030 PRINT" [CTRL] e [RVS ON] [SPACE]
      [CRSR ⇑⇓] [SHIFT] e [CRSR ⇐⇒] ";
2040 NEXT JJ
2050 RETURN
3000 REM DRAW LINE DIAGONALLY UPWARDS
      AND RIGHT FROM X1,Y1
3010 GOSUB 500:REM POSITION CURSOR
3020 FOR KK=1 TO N1
3030 PRINT" [CTRL] e [RVS ON] [SHIFT]
      e [£] [CTRL] e [RVS OFF] [SHIFT]
      e [£] [SHIFT] e [CRSR ⇑⇓] [SHIFT]
      e [CRSR ⇐⇒] ";
3040 NEXT KK
3050 RETURN
4000 REM DRAW LINE DIAGONALLY DOWN-
```

```
      WARDS AND RIGHT FROM X1,Y1
4010 GOSUB 500:REM POSITION CURSOR
4020 FOR KK=1 TO N1
4030 PRINT" [C=] e [*] [CTRL] e
      [RVS ON] [C=] e [*] [CTRL] e
      [RVS OFF] [CRSR ⇑⇓] [SHIFT] e [CRSR ⇐⇒] ";
4040 NEXT KK
4050 RETURN
5000 REM DRAW WINDOW
5010 GOSUB 500:REM POSITION CURSOR
5020 PRINT" [C=] e [A] [SHIFT] e
[*] [C=] e [S] [CRSR ⇑⇓] [SHIFT] e
[CRSR ⇐⇒] 3 volte [SHIFT] e [-] [SPACE]
[SHIFT] e [-] [CRSR ⇑⇓] [SHIFT] e
[CRSR ⇐⇒] 3 volte [C=] e [Z] [SHIFT] e
[*] [C=] e [X] ";
5030 RETURN
6000 REM PAINT ARCHED WINDOW
6010 GOSUB 500
6020 PRINT" [SHIFT] e [N] [SHIFT]
e [M] [CRSR ⇑⇓] [SHIFT] e [CRSR ⇐⇒] 2 volte
[C=] e [+] 2 volte [CRSR ⇑⇓] [SHIFT] e
[CRSR ⇐⇒] 2 volte [C=] e [+] 2 volte [CRSR ⇑⇓]
[SHIFT] e [CRSR ⇐⇒] 2 volte [C=] e
[+] 2 volte ";
6030 RETURN
7000 REM DRAW CROSS
7010 GOSUB 500
7020 PRINT" [CTRL] e [RVS ON] [SPACE]
[CRSR ⇑⇓] [SHIFT] e [CRSR ⇐⇒] 2 volte
[SPACE] 3 volte [CRSR ⇑⇓] [SHIFT] e
[CRSR ⇐⇒] 2 volte [SPACE] [CRSR ⇑⇓] [SHIFT] e
[CRSR ⇐⇒] [SPACE] ";
7030 RETURN
```

UNITA': 19

Esperimento 19.1

```
  10 INPUT"PRIMA FRAZIONE";P,Q
  20 INPUT"SECONDA FRAZIONE"; S,T
  30 A1=P★T+Q★S
  40 B1=Q★T
  50 GOSUB 5500
  60 PRINT"RISULTATO=";C1;"/"; D1
  70 STOP
5500 REM RIDUCI FRAZIONE R1/B1 AI
     MINIMI TERMINI
5510 REM RISULTATO IN C1/D1 LOCALI
     SONO JJ,KK
5520 JJ=A1:KK=B1
5530 IF JJ=KK THEN 5560
5540 IF JJ<KK THEN KK=KK-JJ:GOTO 5530
5550 JJ=JJ-KK:GOTO 5530
5560 C1=A1/JJ:D1=B1/JJ
5570 RETURN
```

Esperimento 19.2B

```
  10 INPUT"PRIMA FRAZIONE";P,Q
  20 INPUT"SECONDA FRAZIONE";S,T
  30 A1=P★T+Q★S
  40 B1=Q★T
  50 GOSUB 5500
  60 PRINT"RISULTATO=";C1;"/";D1
  70 STOP
5500 REM RIDUCI FRAZIONE A1/B1 AI
     MINIMI TERMINI USANDO DIVISIONE
     NON SOTTRAZIONE
5510 REM RISULTATO IN C1/D1 LOCALI
     SONO JJ,KK,LL
5520 REM ERRORE SE A1 O B1 NON SONO
     NUMERI INTERI O SE B1<1
5530 IF A1=INT(A1) AND B1=INT(B1)AND
     B1>=1 THEN 5550
5540 PRINT"PARAMETRI SBAGLIATI-":PRINT
     A1;B1:STOP
5550 IF A1=0THEN C1=0:D1=1:RETURN
5560 LL=1:IFA1<0THEN LL=-1:A1=-A1
5570 JJ=A1:KK=B1
5580 IFKK=0THEN 5620
5590 IFJJ=0THEN JJ=KK:GOTO5620
5600IFJJ>KK THEN JJ=JJ-INT(JJ/KK)★KK:
     GOTO5580
5610 KK=KK-INT(KK/JJ)★JJ:GOTO 5580
5620 C1=LL★A1/JJ:D1=B1/JJ
5630 RETURN
```

Esperimento 19.3

```
  10 INPUT"TRE NUMERI";A1,B1,C1
  20 GOSUB 1000
  30 PRINT"IL MAGGIORE=";X1
  40 GOTO 10
1000 REM TROVA MAGGIORE DI A1,B1,C1
1010 REM E DAI RISULTATO IN X1
1020 X1=A1
1030 IF X1<B1 THEN X1=B1
1040 IF X1<C1 THEN X1=C1
1050 RETURN
```

Esperimento 19.4

Soluzione nell'Appendice B, BIG LETTERS 64

UNITA': 20

Esperimento 20.1

```
10 DIM W$(100)
20 N=0
30 INPUT"NOME";X$
40 IF X$="ZZZZ" THEN 60
50 N=N+1:W$(N)=X$:GOTO30
60 FOR J=N TO 1 STEP -1
70 PRINT W$(J)
80 NEXT J
90 STOP
```

Esperimento 20.2A

```
  10 DIM T$(40)
  20 FOR J=0 TO 40
  30 READ T$(J)
  40 NEXT J
  50 DATA NIL,I,II,III,IV,V,VI,VII,VIII,IX,X
  60 DATAXI,XII,XIII,XIV,XV,XVI,XVII,XVIII,XIX,XX
  70 DATAXXI,XXII,XXIII,XXIV,XXV,XXVI,XXVII,
     XXVIII,XXIX,XXX
  80 DATAXXXI,XXXII,XXXIII,XXXIV,XXXV,XXXVI,
     XXXVII,XXXVIII,XXXIX,XXXX
 100 PRINT"DATI DUE NUMERI"
 110 INPUT X$,Y$
 120 A1$=X$:GOSUB 1000:X=B1
 130 A1$=Y$:GOSUB 1000:Y=B1
 140 Z=X+Y
 150 IF Z>40 THEN PRINT"I RISULTATI
     SUPERANO CAPACITÀ":STOP
 160 PRINT"SUM =";T$(Z)
 170 STOP
1000 REM CONVERTI A1$ NEL NUMERO
     ROMANO B1
1010 FOR JJ=0 TO 40
1020 IF A1$=T$(JJ) THEN 1050
1030 NEXT JJ
1040 PRINT"NON TROVATA ENTRATA":STOP
1050 B1=JJ
1060 RETURN
```

Esperimento 20.2B1

```
  7 REM SOLUZIONE CON MATRICI
 10 DIM N$(20),T$(20)
 20 FORJ=1 TO 20
```

```
   30 READ N$(J),T$(J)
   40 NEXTJ
   50 INPUT"NOME";X$
   60 FOR J=1 TO 20
   70 IF X$=N$(J) THEN PRINT X$;"HA NUMERO
DI TELEFONO";T$(J): PRINT:GOTO 50
   80 NEXT J
   90 PRINTX$;"NON È NELLA LISTA"
  100 PRINT "TELEFONICA"
  110 GOTO 30
1000 DATA MAXWELL,3398123
1010 DATABOHR,558
1020 DATAEINSTEIN,4073189
1030 DATAVON NEUMANN,777000
1040 DATANEWTON,3074
1050 DATA ZUSE,222
1060 DATAPLANCK,1237543
1070 DATABOYLE,146543
1080 DATABABBAGE,03474
1090 DATAPALACE,5674
1100 DATAPTOLEMY,54863
1110 DATAARISTOTELE,66543
1120 DATAMCCARTHY,47
1130 DATADIJKSTRA,645
1140 DATABERZELIUS,777
1150 DATACHARLES,5543
1160 DATAMENDELEEV,645634
1170 DATATSIOLKOVSKY,645332
1180 DATAARCHIMEDES,2
1190 DATAHOYLE,21352
```

Esperimento 20.2B2

```
    7 REM SOLUZIONE SENZA MATRICI
   10 RESTORE
   20 INPUT"NOME";X$
   30 FORJ=1 TO 20
   40 READ N$,T$
   50 IF N$=X$THEN PRINTX$;"HA NUMERO
      DI TELEFONO";T$:PRINT:GOTO 10
   60 NEXT J
   70 PRINTX$;"NON È NELLA LISTA"
   80 PRINT"TELEFONICA"
   90 GOTO 10
1000 DATA MAXWELL,3398123
1010 DATABOHR,558
1020 DATAEINSTEIN,4073189
1030 DATAVON NEUMANN,777000
1040 DATANEWTON,3074
1050 DATA ZUSE,222
1060 DATAPLANCK,1237543
1070 DATABOYLE,146543
1080 DATABABBAGE,03474
1090 DATAPALACE,5674
1100 DATAPTOLEMY,54863
1110 DATAARISTOTELE,66543
1120 DATAMCCARTHY,47
1130 DATADIJKSTRA,645
1140 DATABERZELIUS,777
1150 DATACHARLES,5543
1160 DATAMENDELEEV,645634
1170 DATATSIOLKOVSKY,645332
1180 DATAARCHIMEDES,2
1190 DATAHOYLE,21352
```

UNITA': 21

Esperimento 21.1A

```
10 INPUT"BATTI UNA STRINGA";X$
20 Y$=""
30 FOR J=1 TO LEN(X$)
40 IF MID$(X$,J,1)="E" THEN 60
50 Y$=Y$+MID$(X$,J,1):GOTO 70
60 Y$=Y$+"O"
70 NEXT J
80 PRINT Y$
90 STOP
```

Esperimento 21.2B

```
10 PRINT" SHIFT e CLR HOME"
20 PRINT" CLR HOME CRSR ↑↓ 6 volte CRSR ⇐⇒ 7 volte";
30 PRINT MID$(TI$,1,2);"/";MID$
   (TI$,3,2);"/";MID$(TI$,5,2)
40 GOTO 20
```

Esperimento 21.1C

```
  10INPUT"NOME PREGO";N1$
  20 GOSUB 4100
  30 PRINT"COGNOME È";Y1$
  40 GOTO 10
4100 REM ESTRAI COGNOME DA N1$ E
     PORTALO IN Y1$(VERS. MIGLIORATA)
4110 JJ=LEN(N1$)
4120 IF JJ=0 THEN Y1$="":RETURN
4130 IF MID$(N1$,JJ,1)<"A" OR MID$
     (N1$,JJ,1)>"Z"THEN JJ=JJ-1:GOTO
     4120
4140 FORKK=JJ TO 1 STEP-1
4150 CC$=MID$(N1$,KK,1)
4160 IF NOT(CC$>="A" AND CC$<="Z" OR
     CC$="-" OR CC$="")THEN 4190
4170 NEXT KK
4180 KK=0
4190 Y1$=MID$(N1$,KK+1,JJ-KK)
4200 RETURN
```

Esperimento 21.2

```
  10FOR J=667TO 677
  20 FOR Y1=0 TO 3
  30 X1=SQR(J)
  40 GOSUB5000
  50 NEXT Y1
  60 PRINT
  70 NEXT J
  80 STOP
5000 REM VISUALIZZA DA X1 A Y1 DECIMALI
5010 IF Y1>0 AND ABS(X1)<=999999999
     THEN GOTO 5050
```

```
5020 X1=X1+0.5
5030 PRINT INT(X1);
5040 RETURN
5050 IF X1<0THEN X1=X1-0.5★10↑-Y1:
     GOTO 5070
5060 X1=X1+0.5★10↑-Y1
5070 NN$=STR$(X1)
5080 FOR PP = 1 TO LEN(NN$)
5090 IF MID$(NN$,PP,1)="." THEN PRINT
     LEFT$(NN$,PP+Y1);:RETURN
5100 NEXT PP
5110 PRINT NN$; ".";
5120 FORJJ=1TO Y1:PRINT"0";:NEXTJJ
5130 RETURN
```

Esperimento 21.3A

```
10 INPUT"STRINGA";N$
20 FORJ=1 TO LEN(N$)
30 IF MID$(N$,J,1)>="A" AND MID$(N$,J,1)
   <="Z" THEN 60
40 NEXT J
50 PRINT"NESSUNA PAROLA":STOP
60 N=VAL(LEFT$(N$,J-1))
70 N$=RIGHT$(N$,LEN(N$)-J+1)
80 PRINT N$,2★N
90 GOTO 10
```

Esperimento 21.3B

```
  10 DIM N1$(10),Q1(10)
  20 INPUT"LISTA";X1$
  30 GOSUB 8000
  40 FORJ=1TOX
  50 PRINT N1$(J),Q1(J)
  60 NEXT J
  70 STOP
8000 REM ANALIZZA LISTA ACQUISTI IN X1$
8010 X=0:JJ=1:LL=LEN(X1$)
8020 GOSUB 8200:REM PRIMA CERCA PRIMA
     CIFRA DI UN NUMERO
8030 IF JJ>LL THEN RETURN:REM SE
     STRINGA TERMINATA
8040 GOSUB 8300:REM ESTRAI NUMERO
     (FORNITO IN NN)
8050 IF JJ>LL THEN 8100:REM SEGNALA
     ERRORE SE STRINGA TERMINATA
8060 X=X+1:Q1(X)=NN:REMSCARTANUMERO
8070 GOSUB 8400:REM ESTRAI PAROLA IN
     S1$.Z1=0 SE PAROLA NON SI TROVA
8080 IF Z1=0THEN 8100
8090 N1$(X)=S1$:GOTO 8020
8100  PRINT"NON  COMPRENDO":X=0:RE-
     TURN
8200 REM CERCA INIZIO NUMERO IN X1$
8210 IF JJ>LL THEN RETURN
8220 CC$=MID$(X1$,JJ,1)
8230 IFCC$<"0" OR CC$>"9"THEN JJ=JJ+1:
     GOTO8210
8240 RETURN
8300 REM ESTRAI NUMERO DA X1$
     INIZIANDO IN JJ, E PORTALI IN NN.
     AVANZA
8310 KK=JJ:JJ=JJ+1
8320 IFJJ > LLTHEN RETURN
8330 CC$=MID$(X1$,JJ,1)
8340 IF CC$>="0" AND CC$<="9" THEN
     JJ=JJ+1:GOTO8320
8350 NN=VAL(MID$(X1$,KK,JJ-KK)): RETURN
8400 REM ESTRAI PAROLA DA X1$
     INIZIANDO DA JJ.PORTALA IN S1$, E
     AUMENTA JJ
8405 REM Z1=0 SE NON TROVA PAROLA
8410 IFJJ>LL THEN Z1=0:RETURN
8420 CC$=MID$(X1$,JJ,1)
8430 IF CC$<"A" OR CC$>"Z" THEN JJ=JJ+1
     :GOTO 8410
8440 KK=JJ:JJ=JJ+1
8450 IF JJ>LLTHEN8480
8460 CC$=MID$(X1$,JJ,1)
8470 IFCC$>="A" AND CC$<="Z"THEN
     JJ=JJ+1:GOTO8450
8480 S1$=MID$(X1$,KK,JJ-KK):Z1=1:RETURN
```


UNITA': 22

Esperimento 22.1

```
  10 DATABAIN,BEAVIS,BOWEY,BURNS
     CLARK,FLEMING
  20 DATAGORDON,GREEN,HOOD,KIDD,
     MCCABE,MAVER
  30 DATAMARSHALL,MILLER,NORTH,PACK,
     PERKINS,REED,ROSE
  40 DATAROSS,SIMPSON,SMITH,SYKES,
     TEDFORD,WEBSTER,WOOD
  50 DIMA$(26)
  60 FORJ=1 TO 26:READA$(J):NEXT J
  70 INPUT"BATTI UN NOME";X$
  80 H1=26:L1=1: GOSUB6000
  90 IFM1=-1THEN PRINTX$;"NON
     TROVATO":GOTO 70
 100 PRINT X$;"TROVATO IN ENTRATA";M1
 110 GOTO 70
6000 REM CERCA LISTA ORDINATA:LISTA A$
6010 IF H1<L1 THEN M1=-1:RETURN
6020 M1=INT(0.5★(H1+L1))
6030 IF X$=A$(M1)THEN RETURN
6040 IF X$<A$(M1)THEN
     H1=M1-1:GOTO6010
6050 L1=M1+1:GOTO6010
```

Esperimento 22.2

```
 10DATAROSS,SIMPSON,SMITH,SYKES,
   TEDFORD,WEBSTER,WOOD
 20 DATAMARSHALL,MILLER,NORTH,PACK,
   PERKINS,REED,ROSE
 30 DATAGORDON,GREEN,HOOD,KIDD,
   MCCABE,MAVER
```

```
  40 DATABAIN,BEAVIS,BOWEY,BURNS,
     CLARK,FLEMING
  50 DIMA1$(26)
  60 FORJ=1 TO 26:READA1$(J):NEXT J
  70 N1=26:GOSUB6500
  80 FOR J=1 TO 26
  90 PRINT A1$(J)
 100 NEXT J
 110 STOP
6500 REM BUBBLE SORT DI N1 CAMPI DA
     A1$(1) FINO A A1$(N1)
6510 SS$="NO"
6520 FOR KK=1 TO N1 -1
6530 IFA1$(KK)>A1$(KK+1)THENDD$=A1$
     (KK):A1$(KK)=A1$(KK+1):A1$(KK+1)=
     DD$:SS$="SI"
6540 NEXT KK
6550 IF SS$="SI"THEN 6510
6560 RETURN
```

Esperimento 22.3A

```
  10 DATAADAMS,27
  20 DATABRIGGS,66
  30 DATACHILVERS,29
  40 DATADALE,38
  50 DATACOLIN,12
  60 DATAMACSNOOT,67
  70 DATAWILSON,96
  80 DATA THOMSON,53
  90 DATAWILMOTT,31
 100 DATABAIN,42
 110 DATAMUNDY,64
 120 DATAKRESTIN,85
 130 DATAMCILDOWIE,10
 140 DATAWRAITH,99
 150 DATAGREEN,72
 160 DATAGREENE,52
 170 DATASHEPHERD,53
 180 DATAHUTCHISON,64
 190 DATABLACK,45
 200 DATABAXTER,1
 210 DATASMYRL,99
 220 DATAFLASHMAN,2
 230 DATAMORRIS,75
 240 DATAELLIS,42
 250 DATAMATTHEWS,66
 260 DATAFOLEY,91
 270 DATACOLLINS,36
 280 DATADANIELS,93
 290 DATAJACKSON,77
 300 DATAPEIRCE,78
 310 DATADEWEY,34
 320 DATAMACGREGOR,15
 330 DATAEASON,69
 340 DATAPARSONS,6
 350 DATATHATCHER,45
 360 DATACLAYMORE,85
 370 DATAO'FLAHERTY,66
 380 DATABUNN,5
 390 DATASULLIVAN,85
 400 DATAGILBERT,41
 500 N1=40:REM NUMBER OF PUPILS
 510 DIM S$(N1),M(N1),A1(N1)
 520 FOR J=1TON1
 530 READ S$(J),M(J)
 540 A1(J)=M(J)
 550 NEXT J
 560 REM ORA RIORDINA VOTI IN A1
 570 GOSUB 6000
 580 REM TROVA LIMITE(TRE QUARTI LISTA)
 590 P=A1 (INT(0.75*N1+1))
 600 PRINT"LIMITE =";P
 610 REM ORA VISUALIZZA STUDENTI CHE
     HANNO PASSATO L'ESAME
 620 FOR J=1 TO N1
 630 IF M(J)>=P THEN PRINT S$(J),M(J)
 640 NEXT J
 650 STOP
6000 REM QUICKSORT:RIORDINA N1
     ELEMENTI DI A1
6010 IFSS=1 THEN6030
6020 DIMSS%(100):SS=1: REM DICHIARA CODA
6030 AA=1:BB=N1:SS%(0)=1:PP=1
6040 XX=AA:YY=BB:ZZ=A1(BB)
6050 IFXX>=YYTHEN6090
6060 IFA1(XX)<=ZZTHENXX=XX+1:GOTO
     6050
6070 IFA1(YY)>=ZZTHENYY=YY-1:GOTO
     6050
6080 DD=A1(YY):A1(YY)=A1(XX):A1(XX)=
     DD:GOTO6050
6090 A1(BB)=A1(XX):A1(XX)=ZZ
6100 IFXX-AA<=1THEN6140
6110 SS%(PP)=XX:SS%(PP+1)=BB:SS%(PP+2)
     =2:PP=PP+3
6120 BB=XX-1:GOTO6040
6130 PP=PP-3:XX=SS%(PP):BB=SS%(PP+1)
6140 IFBB-XX<=1THEN6170
6150 SS%(PP)=3:PP=PP+1:AA=XX+1:GOTO
     6040
6160 PP=PP-1
6170 ONSS%(PP-1)GOTO6180,6130,6160
6180 RETURN
```

Esperimento 22.3B

```
  10 REM PRIMA PARTE PROGRAMMA VITA
  20 DIMX$(9,9),Y$(9,9)
  30 PRINT" SHIFT e CLR HOME BATTI
     PROFILO INIZIALE"
  40 PRINT"CON * E SPAZI,"
  50 PRINT"USANDO COMANDI CURSORE"
  60 PRINT"PER ENTRARE NELLA"
  70 PRINT"SCATOLA. USA RETURN"
  73 PRINT"PER TERMINARE CIASCUNA FILA"
  77 PRINT"(ANCHE QUELLE VUOTE)."
  80 PRINT" 3 spazi C= e A
     SHIFT e * 9 volte C= e S"
  90 FORJ=1 TO 9
 100 PRINT" 3 spazi SHIFT e -
     9 spazi SHIFT e -"
```

```
110 NEXTJ
120 PRINT" 3 spazi [C=] e [Z]
    [SHIFT] e [*] 9 volte [C=] e [X]"
130 PRINT" [CLR HOME] [CRSR ↓] 8 volte"
140 FOR J=1 TO 9
150 T$="":INPUT T$
160 FORK=2TO 10
170 X$(J,K-1)=MID$(T$,K,1)
180 NEXTK,J
190 REM VISUALIZZA POSIZIONE CORRENTE
200 PRINT" [SHIFT] e [CLR HOME] [CRSR ↓]
    3 volte 3 spazi [C=] e [A] [SHIFT]
    e [*] 9 volte [C=] e [S]"
210 FORJ=1 TO 9
220 PRINT" 3 spazi [SHIFT] e [-]";
230 FORK=1TO9:PRINTX$(J,K);:NEXTK
240 PRINT" [SHIFT] e [-]"
250 NEXTJ
260 PRINT" 3 spazi [C=] e [Z]
[SHIFT] e [*] 9 volte [C=] e [X]"
270 FOR J=2 TO 8
280 FOR K=2 TO 8
290 T=0
300 IFX$(J-1,K-1)="*"THEN T=T+1
310 IFX$(J-1,K)="*"THEN T=T+1
320 IFX$(J-1,K+1)="*"THEN T=T+1
330 IFX$(J,K-1)="*"THEN T=T+1
340 IFX$(J,K+1)="*"THEN T=T+1
350 IFX$(J+1,K-1)="*"THEN T=T+1
360 IFX$(J+1,K)="*"THEN T=T+1
370 IFX$(J+1,K+1)="*"THEN T=T+1
380 Y$(J,K)=X$(J,K)
390 IF T<2 OR T>3THEN Y$(J,K)=" "
400 IF T=3THENY$(J,K)="*"
410 NEXT K,J
420 FORJ=2 TO 8
430 FORK=2 TO 8
440 X$(J,K)=Y$(J,K)
450 NEXT K,J
460 GOTO190
```

UNITA': 24

Esperimento 24.1

```
10 PRINT" [SHIFT] and [CLR HOME] ";
20 X=20:Y=12
30 POKE 1024+40*Y+X,160
40 POKE 55296+40*Y+X,0
50 GET A$:IF A$=" "THEN50
60 IF ASC(A$)<> 133 THEN 80
70 IF Y>0THEN Y=Y-1:GOTO30
80 IFASC(A$)<>134 THEN 100
90 IF X<39 THEN X=X+1:GOTO30
100 IF ASC(A$)<>135THEN120
110 IF Y<24 THEN Y=Y+1:GOTO30
120 IF ASC(A$)<>136THEN GOTO50
130 IF X>0THEN X=X-1:GOTO30
140 GOTO50
```

Esperimento 24.2A

```
10 DEF FNA(X)=X↑3+(X+7)↑2-100
50 FOR J=2TO3STEP0.1
60 PRINT J;FNA(J)
70 NEXT J
80 STOP
100 REM MIGLIOR STIMA PER SOLUZIONE
    = CIRCA 2.33
```

Esperimento 24.2B

```
10 DEF FNA(X)=X↑3+(X+7)↑2-100
20 DEF FNB(X)=3*X↑2+2*(X+7)
30 X=2
40 Y=FNA(X)
50 PRINT "X=";X
60 PRINT"Y=";Y
70 IF ABS(Y)>0.0000001 THEN X=X-Y/FNB
   (X):GOTO 40
80 PRINT"SOLUZIONE=";X
90 STOP
```

Esperimento 24.3A

Le soluzioni sono indicate come programmi

EXPT 24.3A (T)

e

EXPT 24.3B (T)

se si sta usando una cassetta a nastro o

EXPT 24.3A (D)

e

EXPT 24.3B (D)

per il dischetto.

Esperimento 24.4

Le soluzioni sono date come programmi

DATEPROG (T)

(per la cassetta a nastro) oppure

DATEPROG (D)

per il dischetto.

INDICE ALFABETICO

Altezza 309,314-15
Analisi del denaro 157-160
AND 169,253
Animazione 244
Appuntamenti col computer 269
Argomento 203
Armonia 315-317
Arrotondamento di numeri 212
Attacco e decadimento 311
Bubble Sort 222-223
Bit 233-237
Byte 233-237
BUS seriale 265
Catasta operativa 176,181
Cassetta, uso della 262-265
Codice Morse 268-269
Codici ASCII 254-259
Codici dello schermo 238-240
Comandi DATA 157-163
Comandi del cursore 180
Comando CLOSE 263-267
Comando DEF 260-261
Comando END 260
Comando GOSUB 176-182
Comando IF....THEN 166,253
Comando INPUT 158
Comando INPUT# 263,264,267
Comando OPEN 263-268
Comando PEEK 233,236-237,243-244
Comando POKE 233,238-244
Comando PRINT# 263,266
Comando READ 157-163
Comando RESTORE 157,160
Complessità dei programmi 165-172
Concatenamento 208
Condizione composta 169-171,253-254
Controllo degli Sprites 291
Convenzioni per la
denominazione di subroutine 191
Conversioni di numeri in stringhe 212
Conversione di stringhe in numeri 210
Creazione di quiz
Definizione di caratteri 161-163
Disegno degli Sprites 242
Disegno di programmi 289-290
Due punti(:) 273-285
Durata, nota 166
Eccedenza di parole sullo schermo 309,315
Elementi di matrici 214
Estrazione di cognomi 195-197
File sequenziali 203-205
Forme d'onda 266-268
Frasi casuali 311,315
Frequnze 273-280
Funzione ASC 311,313-315
Funzione CHR$ 256-259
Funzione FRE 257
225
Funzione INT 157
Funzione LEFT$ 206
Funzione LEN$ 203
Funzione MID$ 203
Funzione RIGHT$ 206
Funzione RND 274
Funzione STR$ 212
Funzione VAL 210
Funzione stringa 203-217
GET# 264
Giochi d'avventura 281-285
Giochi di labirinto 281-285
Gioco delle vespe 244-251
Gico Lifestart 229
Grammatica a linea tranviaria 273-277
Indici 195
Indirizzo di concatenamento 176,181
Indirizzo di ritorno 176
Indirizzo secondario 265
Istogramma 190
Istruzioni DATA 160
Istruzioni DIM(ension) 195
Iterazione 158,177,180,197
Jiffy 244
Kernal 236
Leggi di De Morgan 171
Mappa della RAM dei colori 238,241
Mappa della RAM dello schermo 238,241
Matrici 195-200,219-225
Matrici bidimensionali 226-230
Matrici, indici 195-196
Memoria 225-226,235-236
Memorizzazione su cassetta 262
Memorizzazione su dischetto 266
Microprocessore 234-236
Musica 309-317
Newton-Raphson 261
NOT 171
Numero canale 265
Numero di periferica 265
ON 260
Operatori logici 169-172,253-254
AND 169,253
NOT 171,253
OR 170,253
Parametri, subroutine 178
Permutazioni 206-208
Pixel 291
Posizionamento del cursore 206
Probabilità 274
Programma di gestione 187
Programmi per gli Sprites 295-304
Quicksort 224-225
Raccolta di rifiuti 226
RAM 234
Registri 234
Registri delle voci 315
Registri di controllo
periferiche 234-238
Registri di volume 315
Ricerca 219-222
Ricerca dicotomica 220
Richiamo da cassetta 262
Rimozione di lettere da
una stringa 208
Riordino 219,222-225
Risposte 326-335

Robustezza di una subroutine 188
ROM 234
ROM dei caratteri 236-238
Set dei caratteri 1 e 2 239-240,242
Spazio d'indirizzo 233-234
Specifica, subroutine 185
Sprite Editor 289-290
Sprite multicolori 300-301
Sprites 287-304
ST 264,268
Subroutine 175-182,185-193
TAB 228
Tabelle 198
Tasti di funzione 259
TI 311
Timbro 309,315
Unità di controllo disco 266-267
Unità di controllo suono (sintetizzatore) 266-267 234,309
Unità di controllo video 234-238,291-304
Uso del dischetto 265-270
Variabili, input 185
Variabili, matrice 195
Variabili, output 185
Variabili, parametro 178
Visualizzazione della data 161
Voti degli esami 228

INTRODUZIONE AL BASIC PARTE 2

NASTRO 1

UNIT16QUIZ64
UNIT17PROG64
PICTURE64
BIGLETTERS64
UNIT20QUIZ64
UNIT21QUIZ64
QUICKSORT
LIFESTART
MONDRIAN
TONGUE
WASPS64
MAKENAMES(T)
GRAFFS
DUNGEON
SDP
MONSPR
LINEAR
CIRCULAR1
CIRCULAR2
BOUNCE
BUS

NASTRO 2

COSPRED
SOLAR
GROTTY
SPRMAC
PLANETS
DUBLIN
TUNE PLAYER
SHEBA
LIBRARY
EXPT.24.3A(T)
EXPT.24.3B(T)
DATEPROG(T)

NOTE: Tutti i programmi sono registrati in tutti i due lati del nastro.